#178 May 2005

www.circuitcellar.com

CIRCUIT CELLAR®

THE MAGAZINE FOR COMPUTER APPLICATIONS

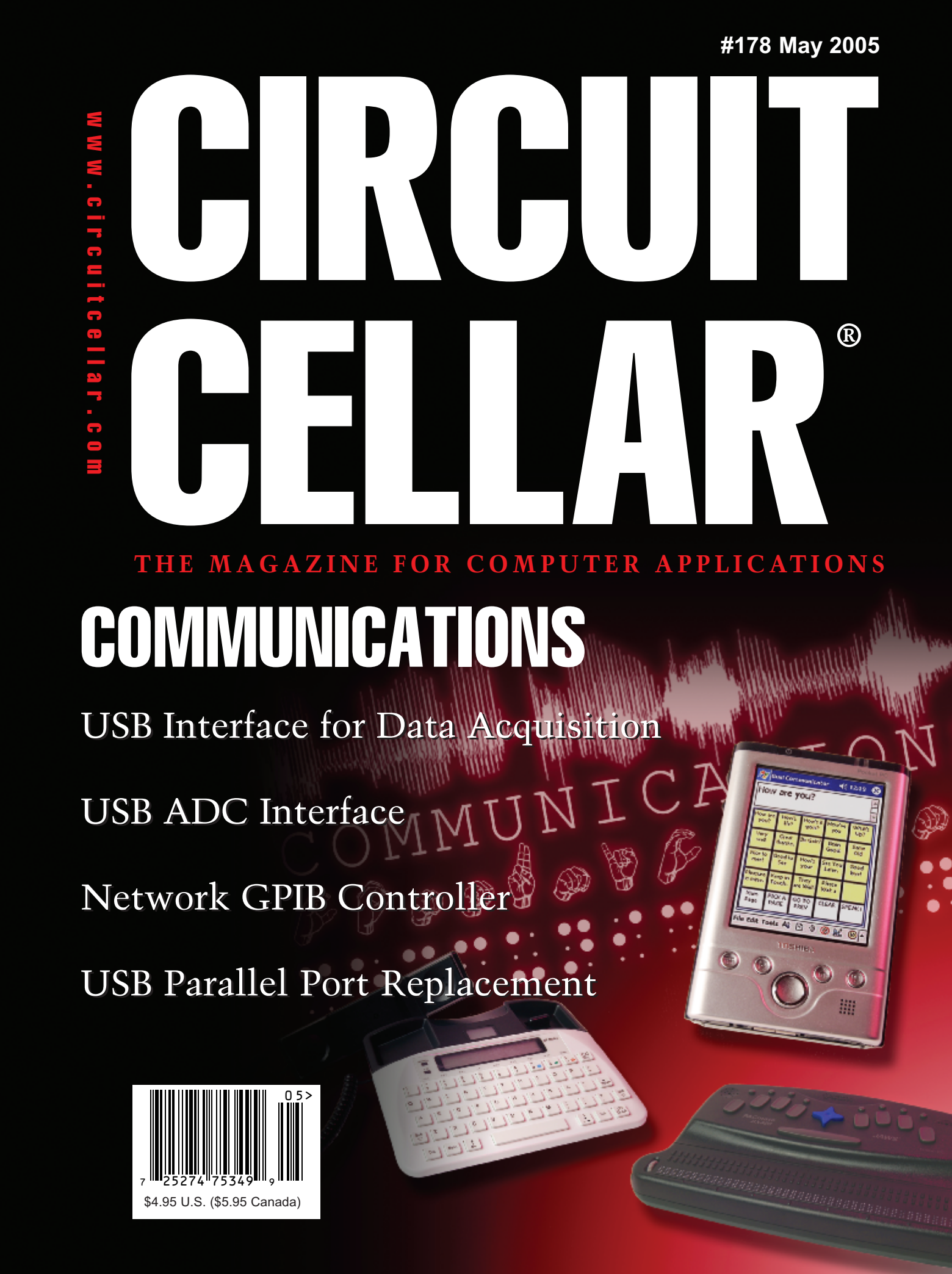
COMMUNICATIONS

USB Interface for Data Acquisition

USB ADC Interface

Network GPIB Controller

USB Parallel Port Replacement



05>
7 25274 75349 9
$4.95 U.S. ($5.95 Canada)

NetBurner Mod 5270

Do the Math!

95 MHz

+95 MIPs

+32 bits

$59 *Priced per 1000*

NetBurner

Networking in 1 day!

1-800-695-6828 • www.netburner.com

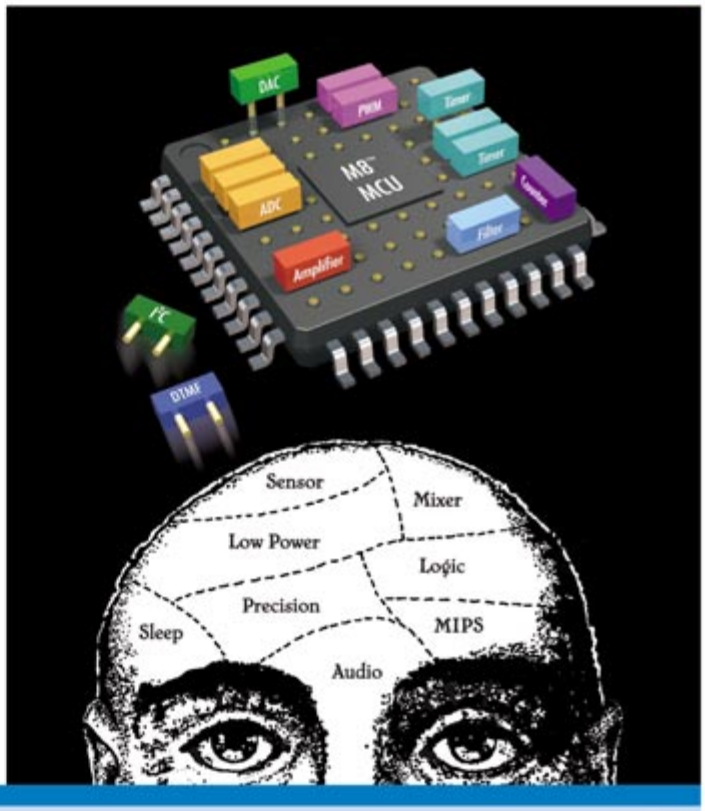
30% OFF
All PSoC Items in Cypress Online Store
www.cypress.com/ad/psoc-cea1

PSoC™ Mixed-Signal Array.
It ll change the way you think about embedded design.

Now with instrumentation-quality programmable analog.

Powerful new programmable analog and digital blocks with memory and MCU for less than $2. Winner of the EDN Innovation of the Year award, our PSoC™ Programmable System-on-Chip™ device is changing the face of embedded design.

- Replace 1,000s of fixed-function devices with a couple of keystrokes
- Dynamically reconfigure a PSoC device, changing functionality on the fly in any application
- Select from hundreds of predefined blocks in our mixed-signal library
- Already designed into 1,000s of applications; check out our online app note library



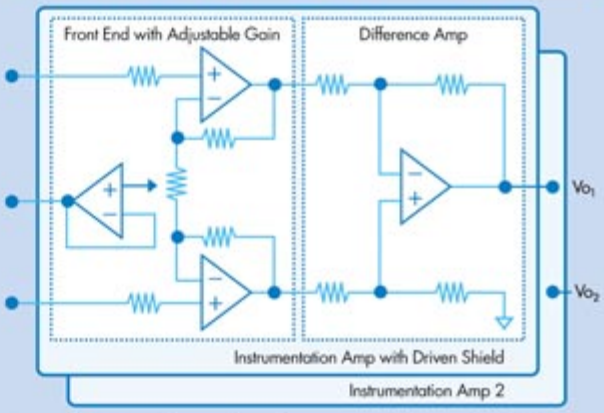
Reduce board size and BOM up to 80%

PSoC Mixed-Signal Arrays with M8 Microcontroller

	CY8C27X	CY8C24X	CY8C22X
CEA ANALOG BLOCKS	12	6	3
DIGITAL BLOCKS	8	4	4
HI REL SONOS FLASH	16K	4K	2K
SRAM	256	256	256
COST	low as $1.99	low as 99¢	low as 69¢

CYPRESS ENHANCED ANALOG™ (CEA™)

• Rail-to-rail analog • Instrumentation amps • Lower voltage offset
• Lower input leakage currents • Programmable gains • Better stability



One of 1,000s of examples of programmable analog blocks.

Check out our free online training and our 4-hour applications support:
www.cypress.com/ad/psoc-cea1

CYPRESS MICROSYSTEMS
A Subsidiary of Cypress Semiconductor

©Cypress Microsystems, Inc. 2003. PSoC, Programmable System-on-Chip, Cypress Enhanced Analog and CEA are trademarks of Cypress Microsystems, Inc. All other trademarks are the property of their respective owners.

FOR PRINTED CIRCUIT BOARDS

Up to 100 pieces

24-72 hour turns

Stencils & Assembly options

Now Up to 100 Pieces!

FOR PROTOTYPE CIRCUIT BOARDS

LEADING INTERNET SUPPLIER

PCBexpress

PCBexpress.com

PROTOTYPE CIRCUIT BOARDS

Successfully selling online since 1997.
Experience you can trust! Try us out today at:
www.PCBexpress.com/cc

Enter the link for a special Circuit Cellar discount!

And for needs beyond PCBexpress...

More options & features

Longer lead times

Production pricing

For protype & production circuit boards (Beyond PCBexpress)

Expect the best.
Easy. Fast. Reliable.

1st order discount

Easiest site to quote & order

Real-time online price matrix
(No signup required to quote online)

PCBpro.com

PCBpro
http://www.pcbpro.com/cc

Seetron Serial LCDs

...almost too easy!

LCDs have a well-deserved reputation for being a pain in the neck to interface. Their non-standard parallel busses and weird timing are enough to turn your hair gray—if you don't pull it out first. Our LCDs have an easy-to-program serial interface (up to 9600 bps) that makes them a snap to use, particularly with friendly micros like BASIC Stamps®.

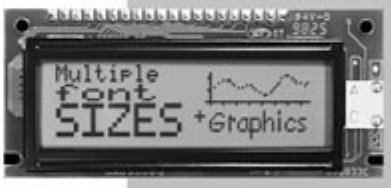
• *Easy hookup and mounting.* We don't stop at simplifying display interfacing and programming. For easy, quick connections, we offer a variety of ready-made wiring harnesses. Need mounting hardware? We've got that, too.

• *In stock right now.* Hate to wait? Our products are in stock and ready to ship the day you place your order.

• *Online satisfaction.* Our customers love our web site—it's fast, uncluttered, and jammed with tech data on our products. The online store runs weekly sales with savings up to 40%. Checkout is a breeze with a simple one-page form.

BPI-216N
2x16 serial LCD
ultra-low power
45.00

ILM-216L
2x16 serial LCD
w/ switch inputs,
config EEPROM
49.00



SGX-120L
120x32-pixel
serial graphics LCD
99.00

TRM-425L
4x20 serial LCD
w/ keypad input,
95-screen memory
119.00

www.seetron.com

Many more models available at seetron.com

Scott Edwards Electronics, Inc.
1939 S. Frontage Rd. #F, Sierra Vista, AZ 85635
phone 520-459-4802 • fax 520-459-0623
web www.seetron.com • email sales@seetron.com

TASK MANAGER

USB Bonanza

If you want to learn more about the Universal Serial Bus, this is your month. I'm dubbing the May issue the "USB Bonanza," and it couldn't have come at a more opportune moment. The lineup includes three fantastic articles about dealing with the inevitable loss of serial and parallel ports on PCs and laptops. If you haven't adapted to the changeover to USB yet, you will likely have to in the near future as more and more manufacturers move in that direction. These informative articles will help you get started.

First up we have Robert Lang's "USB Interface Development" (p. 12). Faced with the elimination of classic serial and parallel ports on PCs, Robert recognized the need to develop an interface to work with USB ports in order to facilitate data acquisition. When he was finished designing his new interface, he thought other people in a similar position would find his experience useful. We agreed. The first step is to familiarize yourself with USB. In this article, Robert provides a helpful overview of USB and walks you through the process of choosing the right chip and software development system to build your own circuit board. If you're in the market for a USB-capable chip, you'll appreciate Robert's thorough examination of today's popular options.

Daniel Cross-Cole also realized the pressing need for a USB interface (p. 30). Daniel's chief goal was to design an interface to support analog-to-digital conversion and measurement applications. He accomplished his goal with a National Semiconductor ADC0820 chip and a Cypress EZ-USB kit. The result is a well-designed interface suitable for a variety of practical applications and classroom experiments. In "USB 2.0 Interface," Daniel shares the valuable lessons he learned along the way and outlines how this interface can be used to teach engineering students about signal processing, among other things.

In addition to Robert and Daniel's articles, we also have a feature about USB by Stuart Ball (p. 76). Stuart's particular concern was losing the parallel printer port he had traditionally used to control the various devices he connects to his PC. He solved this dilemma with the custom-built Atmel ATmega8515 microprocessor-based USBLab, which plugs directly into a USB port. The ATmega8515 is connected to a DLP-USB245 USB adapter by DLP Design. With eight bidirectional data bits, three address bits, a read/write direction control line, a data strobe, and serial input and output, the USBLab has everything you need to overcome the loss of a printer port. The USBLab provides useful debugging capability for devices connected to a PC. In addition to step-by-step instructions on how to design your own USBLab, Stuart includes a helpful sidebar about attaching USB devices to PCs running Windows.

As you contemplate how you'll adjust to the push toward USB, these articles will serve as a guide to help you choose the right course of action and avoid certain pitfalls. Good luck!

jennifer.huber@circuitcellar.com

CIRCUIT CELLAR®

THE MAGAZINE FOR COMPUTER APPLICATIONS

EDITORIAL DIRECTOR/FOUNDER
Steve Ciarcia

MANAGING EDITOR
Jennifer Huber

TECHNICAL EDITOR
C.J. Abate

WEST COAST EDITOR
Tom Cantrell

CONTRIBUTING EDITORS
Ingo Cyliax
Fred Eady
George Martin
George Novacek
Jeff Bachiochi

NEW PRODUCTS EDITOR
John Gorsky

PROJECT EDITORS
Steve Bedford
Ken Davidson
David Tweed

CHIEF FINANCIAL OFFICER
Jeannette Ciarcia

CUSTOMER SERVICE
Elaine Johnston

CONTROLLER
Jeff Yanco

ART DIRECTOR
KC Prescott

GRAPHIC DESIGNER
Mary Turek

STAFF ENGINEER
John Gorsky

QUIZ COORDINATOR
David Tweed

ADVERTISING

PUBLISHER
Dan Rodrigues E-mail: dan@circuitcellar.com

ASSOCIATE PUBLISHER/DIRECTOR OF SALES
Sean Donnelly Fax: (860) 871-0411
(860) 872-3064 E-mail: sean@circuitcellar.com
Cell phone: (860) 930-4326

ADVERTISING REPRESENTATIVE
Shannon Barraclough Fax: (860) 871-0411
(860) 872-3064 E-mail: shannon@circuitcellar.com

ADVERTISING COORDINATOR
Valerie Luster Fax: (860) 871-0411
(860) 875-2199 E-mail: val.luster@circuitcellar.com

ADVERTISING ASSISTANT
Deborah Lavoie Fax: (860) 871-0411
(860) 875-2199 E-mail: debbie.lavoie@circuitcellar.com

Cover photograph Chris Rakoczy—Rakoczy Photography
www.rakoczyphoto.com

PRINTED IN THE UNITED STATES

CONTACTING CIRCUIT CELLAR

SUBSCRIPTIONS:
INFORMATION: www.circuitcellar.com or subscribe@circuitcellar.com
To Subscribe: (800) 269-6301, www.circuitcellar.com/subscribe.htm, or subscribe@circuitcellar.com
PROBLEMS: subscribe@circuitcellar.com

GENERAL INFORMATION:
TELEPHONE: (860) 875-2199 Fax: (860) 871-0411
INTERNET: info@circuitcellar.com, editor@circuitcellar.com, or www.circuitcellar.com
EDITORIAL OFFICES: Editor, Circuit Cellar, 4 Park St., Vernon, CT 06066
NEW PRODUCTS: New Products, Circuit Cellar, 4 Park St., Vernon, CT 06066
newproducts@circuitcellar.com

AUTHOR CONTACT:
E-MAIL: Author addresses (when available) are included at the end of each article

For information on authorized reprints of articles, contact Jeannette Ciarcia (860) 875-2199 or e-mail jciarcia@circuitcellar.com.

CIRCUIT CELLAR®, THE MAGAZINE FOR COMPUTER APPLICATIONS (ISSN 1528-0608) and Circuit Cellar Online are published monthly by Circuit Cellar Incorporated, 4 Park Street, Suite 20, Vernon, CT 06066 (860) 875-2751. Periodical rates paid at Vernon, CT and additional offices. **One-year (12 issues) subscription rate USA and possessions $21.95, Canada/Mexico $31.95, all other countries $49.95. Two-year (24 issues) subscription rate USA and possessions $39.95, Canada/Mexico $55, all other countries $85.** All subscription orders payable in U.S. funds only via VISA, MasterCard, international postal money order, or check drawn on U.S. bank.

Direct subscription orders and subscription-related questions to Circuit Cellar Subscriptions, P.O. Box 5650, Hanover, NH 03755-5650 or call (800) 269-6301.

Postmaster: Send address changes to Circuit Cellar, Circulation Dept., P.O. Box 5650, Hanover, NH 03755-5650.

Circuit Cellar® makes no warranties and assumes no responsibility or liability of any kind for errors in these programs or schematics or for the consequences of any such errors. Furthermore, because of possible variation in the quality and condition of materials and workmanship of reader-assembled projects, Circuit Cellar® disclaims any responsibility for the safe and proper function of reader-assembled projects based upon or from plans, descriptions, or information published by Circuit Cellar®.

The information provided by Circuit Cellar® is for educational purposes. Circuit Cellar® makes no claims or warrants that readers have a right to build things based upon these ideas under patent or other relevant intellectual property law in their jurisdiction, or that readers have a right to construct or operate any of the devices described herein under the relevant patent or other intellectual property law of the reader's jurisdiction. The reader assumes any risk of infringement liability for constructing or operating such devices.

Entire contents copyright © 2004 by Circuit Cellar Incorporated. All rights reserved. Circuit Cellar and Circuit Cellar INK are registered trademarks of Circuit Cellar Inc. Reproduction of this publication in whole or in part without written consent from Circuit Cellar Inc. is prohibited.

Let us make your transition to a lead-free world a risk-free process

The Sierra Express
Work It
Lead-Free Deadline Looms
Prototype shop leads the way... the future of Printed Circuits is NOW!
The Sierra Express
Work It
Lead-Free Deadline Looms

In 2003, the European Union passed legislation stipulating that all printed circuit boards and electronic assemblies must be lead-free by July 1, 2006.*

In the same year, the State of California adopted regulations restricting the import, manufacture, or sale of lead-containing electronics, especially PCBs.

And with Japan, Korea and China all considering similar initiatives, this means a dramatic risk for designers, fabricators and OEMs.

Sierra Proto Express **High-Reliability Lead-Free PCBs**

• RoHS and WEEE compliant • Made using Pb-free laminate with a Td of 364° C
• Capable of withstanding 2000 thermal shocks[1] • Guaranteed reliability
• Quick-turn and/or volume-order • Custom-made to fit your needs

Sierra Proto Express
LEAD FREE

For orders & information, call **1.800.763.7503 x500**
or visit our website: **www.protoexpress.com**

Because to be the best QUALITY LEADER SIERRA PROTO EXPRESS SINCE 1986 ***you need the best!***

Sierra Proto Express, 1108 West Evelyn Avenue, Sunnyvale, California 94086, U.S.A.

*The RoHS (Restriction of Hazardous Substances) and WEEE (Waste from Electrical and Electronic Equipment) initiatives require a complete elimination of lead (Pb) and a reduction of other heavy metals, such as mercury (Hg), cadmium (Cd) and chromium (Cr-IV), in all printed circuit boards and electronic assemblies.

[1]Testing Services courtesy IRTS, manufacturer of the HATS™ Tester. Information about the Highly Accelerated Thermal Shock HATS™ Tester can be found at www.HATS-Tester.com.

May 2005: Communications

FEATURES

12 **USB Interface Development**
Robert Lang

22 **'Net Radio**
Build an Internet Radio Receiver
Bertrand Achard
Atmel AVR 2004 Design Contest Winner

30 **USB 2.0 Interface**
Daniel Cross-Cole

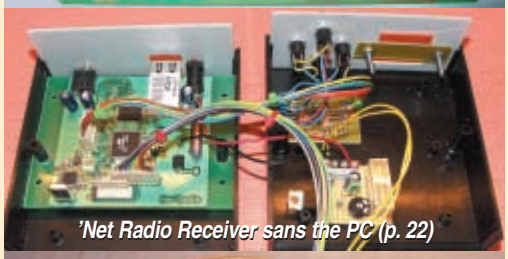
43 **Digital RC Servo Controller (Part 2)**
Circuitry Details
Eric Gagnon

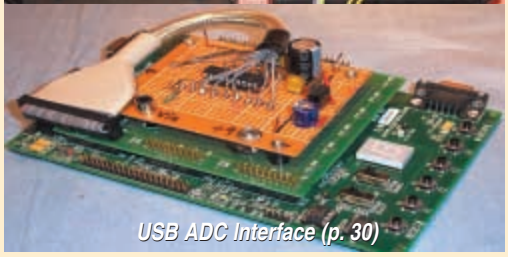
51 **Three-Axis Stepper Motor Controller (Part 2)**
Software Implementation
Prajakta Deshpande & Praveen Deshpande

62 **Network GPIB Controller**
Ron Battles, Patrick Jackson, & Scott Shumate

76 **Connect with USBLab**
Stuart Ball

USB Interface for Data Acquisition (p. 12)


'Net Radio Receiver sans the PC (p. 22)

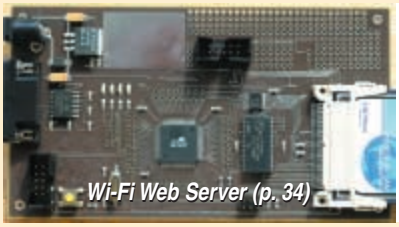

USB ADC Interface (p. 30)

COLUMNS

34 **APPLIED PCs**
Build a Wi-Fi Web Server
Fred Eady

70 **FROM THE BENCH**
Speech Synthesis with SpeakJet
Jeff Bachiochi

80 **SILICON UPDATE**
More Flash, Less Cash
Tom Cantrell


Wi-Fi Web Server (p. 34)

Low-Cost Speech Synthesis (p. 70)

DEPARTMENTS

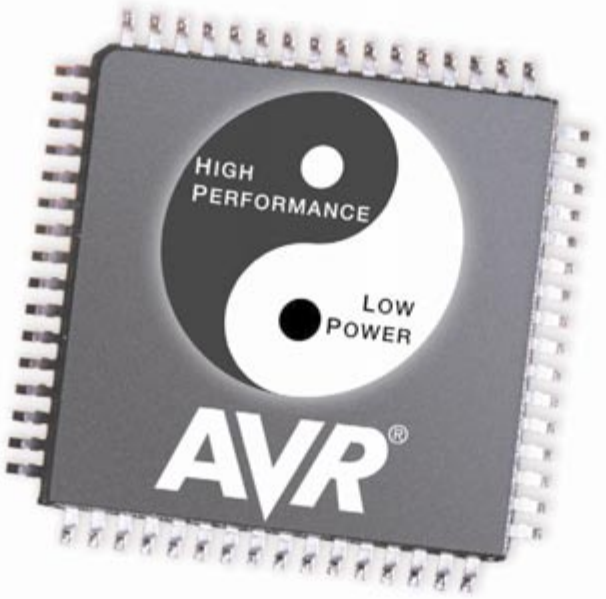
4 **TASK MANAGER**
USB Bonanza
Jennifer Huber

8 **NEW PRODUCT NEWS**
edited by *John Gorsky*

11 **TEST YOUR EQ**
edited by *David Tweed*

94 **INDEX OF ADVERTISERS**
June Preview

96 **PRIORITY INTERRUPT**
Are You Driven by the Means of the End?
Steve Ciarcia



AVR microcontrollers
The perfect combination of performance and power consumption

Specifically designed for the latest battery operated portable applications, Atmel's new Low Power AVR microcontrollers provide six low-power operation modes. These devices work down to 1.8 volts providing maximum battery life. In low power modes such as standby, AVR turns off all internal peripherals and the core. Standby mode leaves only the external crystal running to deliver start-up times as low as 6 clock cycles.

AVR is a RISC CPU that executes instructions in a single clock cycle, with a rich CISC-like instruction set and 32 working registers. AVR microcontrollers have a very high code density and extremely fast execution speeds up to 24 MIPS. The Flash program memory and on-chip EEPROM will help you slash months off your development efforts – save thousands of dollars in project costs, and save YOUR energy as well.

AVR comes in a full selection of package and performance options, and is supported by a full compliment of integrated tools to make your development process fast, simple and successful. Atmel dedicated staff as well as certified technical consultants are available to help and support your projects.

See for yourself: Atmel's Butterfly evaluation kit demonstrates AVR's low power capabilities, and also serves as a development kit for the mega169 microcontroller.

So get your project started off right. Learn more about AVR at www.atmel.com/ad/lowpoweravr, and **register to win a free butterfly evaluation kit.**
See why AVR offers the perfect combination.

Check out AVR today at: www.atmel.com/ad/lowpoweravr

Low Power AVR Typical Power Consumption				
Operating Voltage	Power Down	Real Time Counter	Active 32 kHz	Active 1 MHz
1.8 Volts	Less than 100 nA	4.5 µA	15 µA	240 µA
3 Volts	Less than 250 nA	7 µA	26 µA	420 µA

ATMEL®
Everywhere You Are™

© 2004 Atmel Corporation. Atmel®, AVR® and the Atmel logo are registered trademarks. "Everywhere You Are™" is a service mark of Atmel Corporation.

NEW PRODUCT NEWS

Edited by John Gorsky

USB-POWERED PLED DISPLAY

The **PK202-24-USB** is a 2 × 20 text display with built-in font and provision for up to eight user-defined characters. The 116 mm × 37 mm × 27.5 mm unit comes with 3.2 × 5.55 mm yellow/green characters on a black background with a 160° viewing angle. It sources power and data from one USB cable and accepts USB or serial TTL communication.

Up to 100 modules can be used on the same USB host. Splash and start-up screens can be customized. Luminance is configurable with a timeout setting of up to 90 min. The display can accommodate horizontal or vertical bar graphs and also exhibit double-sized digits. The PK202-24-USB features a keypad interface with 24 keys and a 1-Wire bridge interface that work with 1-Wire devices such as temperature probes. The unit features three low-power GPOs and three high-power advanced GPOs featuring PWM frequency control and a tach sensor.

The PK202-24-USB operates at 0° to 50°C with a supply current of less than 50 mA. Because of its low power consumption and high efficiency, this unit is an excellent choice for LCD or VFD replacement applications.

The display costs **$59** in 1,000-piece quantities.

Matrix Orbital
www.matrixorbital.com

MASTER DEVELOPMENT SYSTEM

The **MDEV-LICAL-MS** Master Development System contains all of the tools necessary to fully explore the capabilities of Linx's new MS Series encoders and decoders. The MS Series is ideal for remote control and command, security, keyless entry, and status monitoring applications. These devices allow the status of up to eight buttons or contacts to be securely transferred via an RF or infrared link.

The MS Series devices have several unique features, including the ability to define user groups and relationships and latched or momentary outputs. They can also assign secure addresses without DIP switches. In addition, the decoder can identify the originating transmitter. Housed in tiny 20-pin SSOP packages, MS Series parts feature low supply voltage and current consumption.

The Master Development System features preassembled evaluation boards complete with Linx RF modules, which allow the encoder and decoder to be quickly tested in a wireless environment. Demonstration software and an on-board USB interface enable connection to a PC. A large prototyping area with breakout headers and a regulated power supply is included.

The Master Development System costs **$249**.

Linx Technologies, Inc.
www.linxtechnologies.com

ETHERNET SBC WITH RESOURCES FOR LINUX

The **WildFire** microcontroller features the resources necessary to run Linux in control and automation applications. With abundant RAM and up to 1 GB of removable SD card program and data storage, the WildFire is well suited for Linux and massive data acquisition.

The versatile WildFire's abundance of digital I/O, analog inputs, and numerous highly programmable timers make it suitable for complex industrial control applications. Its small form factor enables it to be embedded in actuators and tight enclosures.

The Wildfire features 66-MHz ColdFire MPU. It has three serial ports, Ethernet and CAN ports, and enough RAM and flash memory to support full-featured communication stacks and OSs. The SDIO card connector can also hold Wi-Fi or bluetooth cards.

A battery-backed clock calendar hibernation feature restores power to the board after it has turned itself off. This is useful for power-rationed applications such as remote data acquisition.

The WildFire controller costs **$199**. The development kit costs **$399**.

Intec Automation
www.steroidmicros.com

Get a hot deal on ColdHeat

FREE ColdHeat™ Soldering Tool

(a $19.99 value)

With purchase of $20 or more of component parts, soldering equipment, test equipment or Kronus™ Tools.

RadioShack®
You've got questions. We've got answers.®

Must present coupon at time of purchase.

FREE ColdHeat™ Soldering Tool

With purchase of $20 or more of component parts, soldering equipment, test equipment or Kronus Tools.

A $19.99 Value

One coupon per customer.

RadioShack®

CHF222

RSS Instructions: If the customer purchases $20 or more in Component Parts, Test Equipment, Soldering Equipment or Kronus Tools, use the Price Change key to reduce the price of the ColdHeat (640-2102) to free. Select Reason Code "7. Promo/ Coupon" and scan the barcode on the coupon.

Valid at participating stores only. May not be used online or by phone. Excludes services and special orders. May not be combined with certain other discounts. No cash value. No photocopies. Void where prohibited. Offer expires May 31, 2005.

California residents: WARNING: These products contain, or when used for soldering and similar applications produce, chemicals known to the State of California to cause cancer and birth defects (or other reproductive harm).

NEW PRODUCT NEWS

SBC SYSTEM FOR INDUSTRIAL APPLICATIONS

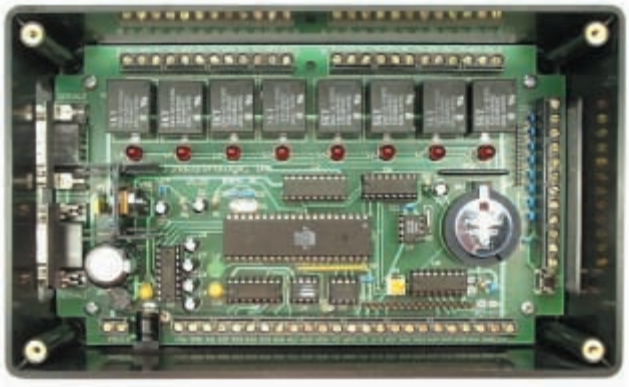
The **IC51** is designed to be a complete industrial controller assembly. It includes a versatile complement of the input and output circuitry needed by many common control applications.

The IC51, which is based on the AT89C51 microcontroller, can be reprogrammed using any number of software development tools and device programmers available for Atmel microcontrollers. You can program the board as a stand-alone controller using its on-board Tiny Machine Basic programming language, or you can use it as an RS-232 serial data acquisition board.

Communication is accomplished via two serial ports with true RS-232 drivers and DE-9 connectors matching PC-compatible serial port pin outs. The IC51 includes a real-time clock/calendar chip with on-board battery backup and registers that are integrated into the variables of the Tiny Machine Basic language.

Convenient screw terminal blocks are used for all of the digital, analog, and relay connections, which include eight 10-A form C relays, eight logic levels signals that you can use as inputs or outputs, eight digital outputs, and eight 12-bit A/D inputs. Its on-board power supply has a wide input range and includes 5- and 12-V outputs at the screw terminals for use by external devices.

The IC51 features LCD support circuitry that interfaces to a 16-pin 0.1? header connector, matching most LCD module connections. It also has an on-board contrast potentiometer for the LCD module. Included in the Tiny Machine Basic language are LCD commands that make it easy to display information on an LCD module.

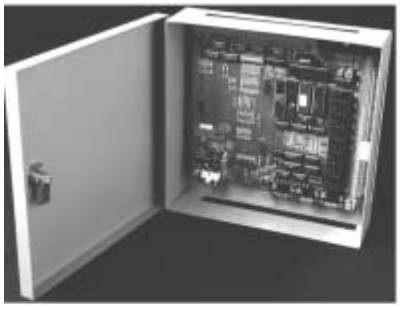


The IC51 package is shipped complete with all the items necessary to begin application development immediately: a serial port cable for connection to a PC, a wall block power supply, host computer software and programming examples, and hardware and software reference manuals.

The IC51 costs **$159**. An enclosure costs **$30**.

IndustroLogic, Inc.
www.industrologic.com

HomeVision-Pro



CONTROLS:
- Lighting
- HVAC
- Security
- A/V
- Much more...

HomeVision-Pro is the newest, most powerful automation controller in the award-winning HomeVision® product line. It's configured from a PC, but can then run standalone. Features:

- Two-way X-10 & IR
- Video screens on a TV
- 8 relays
- 16 digital & 4 analog I/O
- 64 digital temp sensors
- 1 USB & 3 serial ports
- Powerful scheduling (>8000 lines, >2000 objects)
- Clock
- PC web server
- Expandable
- Software included
- 3-year warranty

Custom Solutions, Inc.
800-398-8882, Web: www.csi3.com, E-mail: csi@csi3.com

It gets no easier than this!

SPLat is an innovative new set of solutions that's changing how OEMs world-wide think about embedded controls. With SPLat you can program a quick and dirty sequencer literally in minutes, yet the power is there to make sophisticated state machine based multi-tasking systems with elegant user menus and floating point arithmetic. Our hardware comes with real-world I/O interfaces, ready to wire in, straight out of the box. We have a wide range of special function add-ons for hard to implement tasks like AC current measurement and DC motor control. Best of all, you can do a proof of concept using off-the-shelf product and later migrate all your knowledge and 98% of your code to a fully customized SPLat controller for quantities down to a few hundred. Check out our website today!

Example: SPLat MS12OEM216
- 12 digital I/Os *plus*
- 2x16 LCD, blue backlight
- 5 push buttons
- 4 programmable LEDs
- Expandable

$99 Qty 1

SPLat CONTROLS
Ph. (781) 729-3005

Use the exact URL below for a special offer on your first purchase

www.splatco.com/cc

CIRCUIT CELLAR

Test Your EQ

Edited by David Tweed

Problem 1—What does the following C code compute? Assume that `uint16` is an unsigned 16-bit integer type on this platform. Analyze its operation in mathematical terms.

```
#define Precision  ( 16 >> 1 )
uint16 Function (uint16 Number)
{
   uint16 Result;
   uint16 Result2;
   uint16 Mask;
   uint16 Mask2;
   uint16 Test;
   uint16 Power;
   Result = 0;
   Result2 = 0;
   Mask = 1 << (Precision - 1 );
   Mask2 = 1L << ( ( Precision - 1 ) << 1 );
   for (Power = Precision; Power > 0; –Power) {
      Test = Result2 + Mask2 + ((uint16)Result << Power);
      if (Test <= Number) {
         Result2 = Test;
         Result |= Mask;
      }
      Mask >>= 1;
      Mask2 >>= 2;
   }
   return Result;
}
```

Problem 2—Similarly, what does the following C code compute? Assume that `uint32` is an unsigned 32-bit integer type on this platform. Again, analyze its operation in mathematical terms.

```
uint32 func (uint32 n)
{
  uint32 result = 0, bit, trial;
  bit = (n >= 0x10000) ? 1<<30 : 1<<14;
  do {
     trial = result + bit;
     if (n >= trial) {
       n -= trial;
       result = trial + bit;
     }
     result >>= 1;
     bit >>= 2;
  } while (bit)
  return result;
}
```

Problem 3—What is the signal format used in a digital telephony T1 type of circuit?

Problem 4—What is the analog bandwidth of a T1 signal?

Contributed by David Tweed

What's your EQ?—The answers are posted at www.circuitcellar.com/eq.htm

You may contact the quizmasters at eq@circuitcellar.com

JAMECO
ROBOT STORE
We're Back!
Robotic Arm Kit
(No. 4-523)
Twin Motor Gearbox Kit
(No. 3-709)
Yes! The internet's first and best robotics source has been purchased by Jameco Electronics, and we're committed to greatly expanding the product line quickly!
Each month we'll add hundreds more new robotics products. Plus we'll continue to supply favorites like Muscle Wire®, the OctoBot Survivor™ robot and more.
Be sure to visit us online at
www.RobotStore.com
or call
1.800.374.5764
...and see what's changed!
For All Your Robotic Needs!

USB Interface Development

Need a USB interface for data acquisition? Simply choose a chip and software development system, and then build a circuit board. You can test the board with an uncomplicated HID mouse application.

When I finally got around to buying a laptop computer, I was surprised by its lack of classic serial and parallel ports. The general trend for new PCs is the replacement of classic serial and parallel ports with Universal Serial Bus (USB) ports. Previously, I'd used the classic ports for controlling equipment and data acquisition. But with their elimination, I'd lost my interface capability. Thus was born this USB interface project.

A LOOK AT USB

Several hardware manufacturers got together and developed the USB 2.0 specification, which has a high data transfer rate of 480 Mbps. USB 2.0, which is backward-compatible with the previous USB 1.1 standard, uses the same cables, connectors, and software interfaces. USB 1.1 supports data rates of 1.5 Mbps (low) and 12 Mbps (full).

USB enables you to plug in and unplug peripherals without having to power down your computer. This process, called enumeration, initiates communication with the peripheral and identifies the device driver that should be loaded. Each peripheral is assigned a unique address during enumeration that's used for runtime transfers. During runtime, the host PC initiates transactions to specific peripherals, and each peripheral accepts its transactions and responds accordingly.

A USB peripheral is a slave that obeys a defined protocol. It must react to request transactions sent from the host PC. The peripheral responds to control transactions that, for example, request detailed information about the device and its configuration. The peripheral sends and receives data to and from the host using a standard USB data format.

You can use Microsoft's free USBView utility to display the USB connection tree and show the USB devices connected to it together with their configuration data. You'll find this extremely useful for debugging USB enumeration errors. USBView runs on Windows 98, ME, 2000, and XP. USB ports transfer data through endpoints, which are analogous to buffers.

Photo 1 shows the USBView program's output with my hardware and test software connected. USBView tells you that the device has one configuration and a maximum packet size of 8 bytes. It operates at low speed with one open pipe. There's an interrupt type of endpoint that transmits 3 bytes of data every 10 ms.

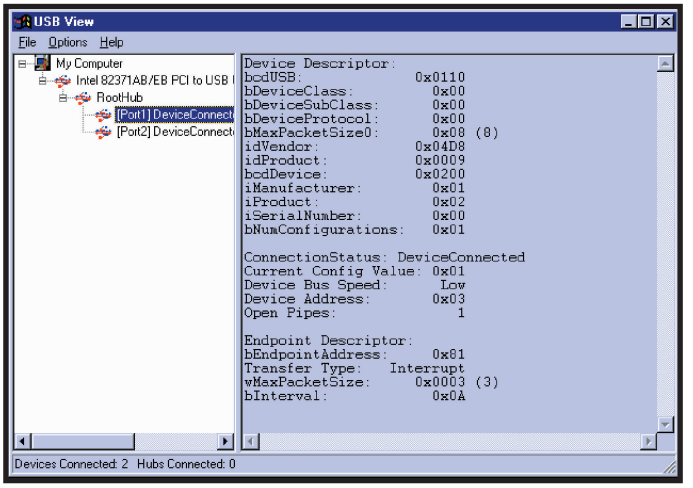


Photo 1—*The output of the USBView program running on Windows 98 shows information about the USB interface.*

CHOOSE A PROCESSOR

I considered several different microprocessors for this project (see Table 1). Each has its own advantages and disadvantages.

The FTDI FT232BM is the simplest chip to use, but it has two disadvantages. The chip manufacturer supplies a simple serial driver with a user application-programming interface. It's extremely simple to write a program in Visual Basic to send USB data out the chip at just about any data rate. The problem is that the Microsoft USB HID driver sends data in data packets. The HID device must decode the data and process it. Because the FT232BM has no on-chip programming capability, it can't be used as an intelligent USB device that does internal processing. It's also available only as a 32-pin SOIC, which makes it a little difficult for the home solderer to use.

The second chip I considered was the Cypress AN2131. The problem with this chip, in addition to its small form factor, is that it has no permanent memory. The AN2131 has USB firmware programming that handles the initial USB response; however, the USB interface also must be used to download programming from the PC to the device before it can act as an intelligent USB device. I decided to avoid the additional complication of downloading

Chip	Driver	API	Software	Program memory	Speed	Form factor	Serial ports	Cost	Comments
FTDI FT232BM	Vendor	Vendor	None	Unknown	6 MHz	32-pin SOIC	1	$5.50	Chip programming not required. Vendor-written API is easy to incorporate in Windows program, but the vendor-written driver and API don't support Microsoft MIDI or HID device drivers. Mainly used for converting USB port to serial port.
Cypress AN2131 (8051 derivative)	Microsoft	Microsoft Windows	C or assembly	8-KB RAM	24 MHz	44-pin PQFP	2	$6.50	EZ-USB core will enumerate it as a USB device without firmware. USB interface can be used to download code to RAM.
Microchip PIC16C745	Microsoft	Microsoft Windows	C or assembly	8-KB OTP or UV erase	24 MHz	40-pin PDIP	1	$5.15–$17.26	Interfaces with existing Microsoft drivers and API. Limited memory options make software development difficult.
Microchip PIC18F2455	Microsoft	Microsoft Windows	C or assembly	24-KB Flash memory	4–48 MHz	28-pin PDIP	1	$7.76–$9.32	Has erasable flash memory.

Table 1—*Take a look at a few of the popular microprocessors with USB capability.*

programming to the device.

The Microchip PIC16C745 was the third chip I considered. This looked extremely promising with a nice 40 PDIP form factor; however, it's only available in a one-time programmable or UV erasable form. Because I didn't feel confident programming the chip correctly the first time, and because I didn't want to purchase a UV eraser, I decided to pass on it.

The final chip I looked at was the brand new Microchip PIC18F2455, which had everything I wanted: a nice 28 PDIP form factor and plenty of electrically erasable flash memory. Because the flash memory isn't lost when the chip is powered down, there's no need to reload the program from the PC at each power-up.

PIC18F2455 DETAILS

The PIC18F2455 is one of Microchip's new USB-enabled flash memory microprocessors. The 24 KB of flash program memory allows you to store about 12,000 16-bit instructions that can be erased and reprogrammed electrically. The flash program memory supports 100,000 write/erase cycles and has a retention period longer than 40 years. The microprocessor has 2048 bytes of RAM.

The PIC18F2455 has many of the features you've come to expect from Microchip: serial communication module (UART), four timers, 10 10-bit A/D channels, and a master synchronous serial port (MSSP), which is useful for communicating with peripheral devices such as serial EEPROMs and supporting SPI and I²C protocols. The microprocessor has 75 base instructions. It offers seven different operating modes for managing power consumption. The lowest power mode, Sleep mode, draws approximately 1 µA.

The USB engine is V.2.0-compliant and operates at low (1.5 Mbps) and full (12 Mbps) speeds. It supports control, interrupt, isochronous and bulk transfers, and up to 32 endpoints. It has 1 KB of RAM starting at 400h that's shared between the CPU and the USB engine.

You can configure the shared memory for optimal use. The first few locations are used for endpoint (buffer) descriptors: BDnSTAT, BDnCNT, BDnADRL, and BDnADRH. The number of endpoints using dual (ping-pong) buffering locations defines the locations. BDnSTAT is the status register for the *n* endpoint. Bit 7 determines who owns the buffer (CPU or SIE) and who can write to it. BDnCNT is the number of bytes the buffer can hold. BDnADRL and BDnADRH are the low- and high-order bytes of the address where the buffer starts in the 400h–7FFh range. The PIC18F2455 supports a dual-buffering scheme in which each endpoint has two buffers defined. Up to a total of 64 buffers can be defined.

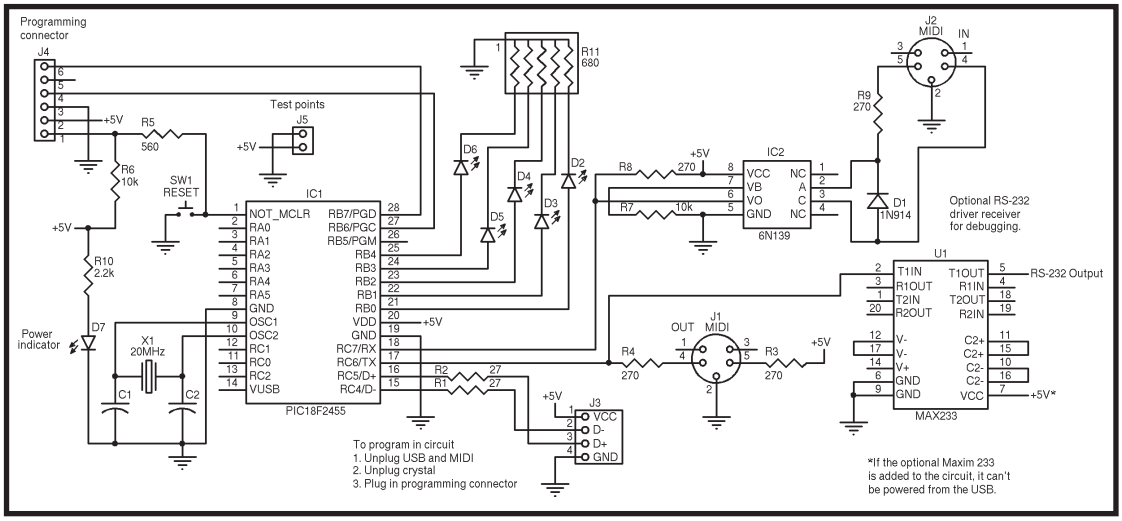


Figure 1—*The schematic for the USB interface includes the optional MIDI and debugging components.*

BUILD THE HARDWARE

The circuit for the USB interface is fairly simple (see Figure 1). There are few enough connections that it can

be built on a 2″ × 4.5″ prototype board with hole spacings at 0.1″. The heart of the system is the PIC18F2455 that handles the USB protocol. Because I planned to eventually develop the device into a USB-MIDI interface, some MIDI plugs and a 6N139 high-speed, high-gain Darlington optoisolator were added, but these aren't needed for the mouse demo in this project.

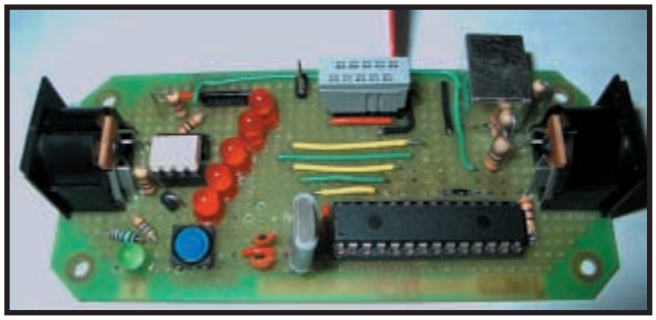


Photo 2—*The completed USB interface circuit board shows the programming connection (top center), the USB connector (top right), and the Reset button (lower left).*

Five LEDs are used to announce the state of the microprocessor and a power LED. A 20-MHz crystal oscillator provides the clock for the system. A manual push button is used to reset the system. There's an ICSP programming connector used to bring programming signals from the programmer to allow for the programming of the chip without removal from the circuit board. A USB connector completes the hardware. The completed circuit board is shown in Photo 2.

For the initial debugging, I added an optional MAX233 RS-232 driver/receiver, which allowed me to send debugging data to be printed by the PC at 38,400 bps. Some USB hubs don't have over-current protection. It's important that you stay under the 100-mA limit in order to prevent damaging the hub. The circuit board draws approximately 30 mA (55 mA with all LEDs illuminated) without the MAX233. If you decide to add the MAX233, it can't be powered from the USB or damage may occur. Heed this voice of experience!

I changed two things in the circuit in Figure 1. I added a 470-nF polarized capacitor that connects pin 14 (VUSB) to ground. This change enables the device to operate at the USB's high or low speed by just toggling bit 2 of the program variable ucfg. Secondly, I connected a 1-µF polarized capacitor between 5 V and ground on the USB connector. This limits the inrush current when the device is plugged into a USB port.

CHOOSING SOFTWARE

Having chosen a chip, the next problem was to determine how to program it. Assembly language and C language were the two obvious choices. After looking at some other USB programs, it appeared that the program would be fairly complex and would probably be a major undertaking in assembly language, so I decided to use C language. The main challenge to using C was locating a compiler that wouldn't completely deplete the budget for the project.

I searched the Internet for a good C compiler (see Table 2). PICC was by far the most complete and fully developed C compiler. At $1,900 it was also the most expensive for a compiler that would work with all Microchip

Features	SourceBoost BOOSTC	HI-TECH PICC	B Knudsen Data CC5X	Custom Computer Services PCWH
Version	1.8	8.05	3.2	3.212
ISO ANSI C	No	Partial, no recursion	No	No
IEEE 32-bit floating point	No (planned for future)	Yes	32 bits but not IEEE	32 bits but not IEEE
Mixed C and assembly	Yes	Yes	Yes	Yes
PDF User manual	No. On-line help for IDE, minimal on-line help for compiler.	Yes (395 pages)	Yes (109 pages)	Yes (257 pages)
Separate linker	LINKER.EXE	HLINK.EXE	MPLINK.EXE	No
Constants in program memory	Yes (strings only)	Yes	Yes	Yes
Environment	Yes	Yes	No	Yes
User-defined variable location	Yes	Yes	Yes	Yes
Array size definition	Yes	Yes	Yes	Yes
Delete unused functions	Yes	Yes	Yes	Yes
Structure/union	Yes	Yes	Yes	Yes
Project support	Yes	Yes	No	Yes
Trial	15-day trial of full version. Time unlimited trial of limited version.	21-day trial of full version. Time unlimited trial of limited version.	Time unlimited trial of limited version	30-day trial of limited version
Download size	7.7 Mb in seven files	6.8 Mb in one file	0.4 Mb in one file	5.2 Mb in one file
ANSI standard libraries	Minimal	Most	Math	Some
Platforms	Windows	DOS	DOS	Windows
Supported processors	Scenix, PIC10–PIC18	PIC10–PIC17 PIC18 requires separate PICC-18 compiler (additional $950)	Microchip PIC10–PIC16 PIC18 requires separate CC8E compiler (additional $250)	PIC10–PIC18
Price	$70 full	$950	$250	$425

Table 2—*These popular C compilers are for Microchip PIC microprocessors.*

Noritake

A PROGRAMMER'S WISH LIST OF BUILT-IN FEATURES

7000 SHOW AND TELL™ series

MORE BUILT-IN FEATURES THAN YOU MAY EVER NEED

- Up to 4 Independent Windows with Individual Control
- Many Int'l Font Sets with 16 User-defined Characters
- Font Magnification
- Horizontal & Vertical Scrolling
- Proportional Fonts
- 8 Levels of Brightness Control
- Built-in Screen Savers
- Easily Combine Text & Graphics on One Screen

VACUUM FLUORESCENT DISPLAY
vfd

And for those needing even more Functions & Power see our New 3000 Series Modules

Established 1904
electronics@noritake.com
Factory ISO 9001 Certified

www.noritake-elec.com

itron

Atlanta	Boston	Chicago	Dallas	Los Angeles	New York	Canada
Toll-Free	Toll-Free	Toll-Free	Toll-Free	Toll-Free	Toll-Free	
1-888-326-3423	1-888-395-3423	1-800-779-5846	1-888-318-3423	1-888-795-3423	1-888-296-3423	1-416-291-2946

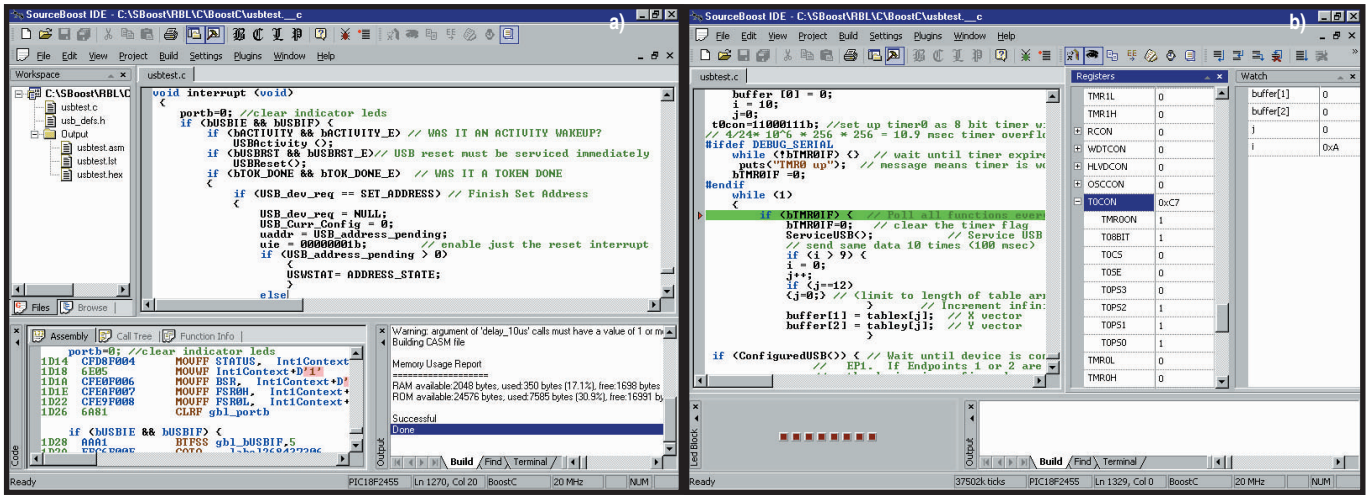


Photo 3a—*This view of the BOOSTC integrated development environment shows the project window, files window, assembly code window, and build output window.* **b—***After compiling, the program can be executed in Debug mode. The green highlighted line shows the position in the source and code windows that corresponds to the current program counter (PC). Other windows show the contents of the processor registers and program variables.*

PICs. I eliminated the $500 CC5X because it didn't have an integrated development environment (IDE). The PCWH on the other hand had a Windows-based IDE, but I eliminated it because of its price. I ended up choosing the most economical option, BOOSTC.

When I downloaded SourceBoost BOOSTC, it was in active ALPHA testing and available for free. But even now the compiler is the most economical. The downloading process for the BOOSTC compiler/IDE was nice because I had a 56k modem and had trouble downloading some of the other large compilers. SourceBoost thoughtfully broke up their 7.7-Mb system into seven 1-Mb files, which were easy to download using even the slowest modem.

COMPILER/IDE

After cranking up the BOOSTC compiler in Windows, I was pleasantly surprised by the easy-to-use user interface (see Photo 3a). The file pane shows the C coding as it's edited or compiled. The code pane shows the original C coding and the generated machine code/assembly language equivalent.

One quirk that I noticed while compiling code for a PIC16F877 was in the machine code listing. The order of the bytes was reversed from what had been produced by the Microchip assembler. The assembler produced a machine code of `0E03` for `SWAPF STATUS, W`, while the BOOSTC produced `030E`, which was the same order as the data in the hex file. Microchip reverses the order of the bytes when it writes the hex file. It's something to be aware of, but the end result, the hex file, is identical in both cases.

The documentation for the BOOSTC compiler is being developed. It's currently available as an on-line HTML file. If you are new to C, check out Ernest Ackermann's book, *The Essentials of the C Programming Language*.

Photo 3b shows the IDE in Debug mode. The file pane shows the C program. The instruction being executed is highlighted. The LED bar displays a port's output. Note that the Register bar shows the content of the processor registers and bits. As you can see, the TMR0ON bit is set in the T0CON processor register. The definitions of bits and registers are controlled in a processor TDF file that can be customized. The Watch window allows you to display any of the program variables.

I encountered several problems as I wrote the program because the compiler isn't ISO-ANSI C-compliant and the documentation was rather limited. I tried converting an ISO ANSI C program to compile with BOOSTC. There were several problems related to compiler errors during program development. The SourceBoost on-line forum was extremely valuable for reporting compiler errors (www.sourceboost.ipbhost.com). The response time from the developers with a compiler fix or workaround was usually one or two days. The on-line documentation for the IDE was adequate. The on-line documentation for the compiler is constantly improving and more coding examples are being added.

PROGRAM THE PROCESSOR

The PIC18F2455 uses a different programming algorithm to write and erase the flash memory than previous PIC18*xxxxx* microprocessors. Because the PIC18F2455 chip was brand new when I got it, third-party programmers had not yet modified their software to support the new algorithm. This created a delay while I wrote some software to use the new algorithm. By now, though, I'm sure IC-PROG and other third-party programming software support the new chips.

TEST SOFTWARE

The first program I wrote with the BOOSTC compiler was a small one to blink the LEDs on the circuit board. The program was relatively simple to write. I also used it to debug the programming software and the ICSP connection.

The next piece of software was based on a USB HID example program from Microchip. The example program set up the microprocessor as a mouse device, and it supplied mouse coordinate data to the PC. It basically caused the mouse pointer to move around in a figure-eight pattern. The Microchip example was written for a different microprocessor and a different C compiler. You may download the source code for the Irritating Mouse test from

Innovation and creativity could win you a share of $20,000*

The Renesas M16C Design Contest 2005 is the perfect opportunity to work with the Renesas M16C family. Your innovation and creativity could win you a share of $20,000 in cash prizes!*

The Renesas M16C™ MCU platform is the industry's broadest microcontroller product line, offering seamless code, peripheral and pin compatibility from small 20-pin to large 144-pin devices, all within a single common software development environment. The complete line of highly optimized MCUs delivers practical solutions for your applications.

For the official rules and entry form, go to www.circuitcellar.com/renesas2005m16c.

Deadline: June 30, 2005

Everywhere you imagine RENESAS

M16C PLATFORM™

DESIGN CONTEST 2005

*See the web site for complete details.

RENESAS CIRCUIT CELLAR®

the *Circuit Cellar* ftp site.

The program uses minimal interrupts. The interrupt routine handles only USB reset interrupts, activity (wake-up), and token-done interrupts. Most of the processing is done on an approximately 10.9-ms timer routine in the main program. The main program begins by flashing the five LEDs on the board. This is just to give feedback that the microprocessor is running.

Next, the main program calls the `UART_Init` routine to initialize the serial port to 38,400 bps for debugging output. This may be confusing, so let me explain. I was using a 20-MHz crystal as the clock for the microprocessor, and I supplied a value to the `UART_Init` routine that sets a data rate of 38,400 for a 24-MHz crystal. This is because a constraint for using the USB at full speed requires the CPU clock to run at 24 MHz. The configuration bits in _CONFIG1L are used to convert the 20 MHz provided by the crystal to 24 MHz for the CPU.

After `InitUSB` is called to initialize the USB port, the `T0CON` register is initialized to start the T0 timer with a 10.9-ms period. Following this, the main program waits until the timer overflow flag is set to call the `ServiceUSB` function to handle any USB requests. The `ServiceUSB` function does one of four things: it restarts the system if a stall is detected; it reports USB errors; it puts the USB to sleep if the IDLEIE bit is set in the UIE register; or it calls `Process_Req` if the processing of a pending transaction is complete.

Listing 1—*The device descriptor is the first USB descriptor sent to the host. The device descriptor begins to define the USB device to the host.*

```
const char DeviceDescriptor [] = {0x12,  // 18 bytes long
0x01,        // Device descriptor type
0x10, 0x01,  // USB specification release (2.00)
0x00,        // Class code
0x00,        // Subclass code
0x00,        // Protocol code
0x08,        // Maximum packet size
0xD8,0x04,   // Vendor id (04d8) microchip
0x09,0x00,   // Product id (0009)
0x00,0x02,   // Device release number 2.00
0x01,        // Index to string that describes vendor
0x02,        // Index to string that describes product
0x00,        // Index to string that describes serial number (none)
0x01         // Number of possible configurations
};
```

The interrupt routine is minimal. It's triggered by USB interrupts. If there's activity on the bus while the device is asleep, the activity interrupt will wake the device up. If a USB reset is received from the PC, it's serviced immediately. And, finally, if there's a set address request from the PC, it's also serviced immediately.

USB DESCRIPTORS

Each USB class has a set of descrip-

tors that are used to give information to the PC about the USB device. Some of the types of descriptors include: device, configuration, string, interface, endpoint, HID, and report.

The device descriptor for this project is shown in Listing 1. The first 2 bytes are pretty standard for all descriptors. The first byte contains the length. The second byte gives the descriptor type. In this case, a 1 indicates that it's a device descriptor. The descriptor gives the maximum packet size and other information.

The vendor ID and product ID are of particular interest. The vendor IDs are assigned by the USB Implementers Forum. Microchip has a vendor ID (04D8), so I'll use it for this experimental device. Vendor IDs and product IDs must be used carefully because they can cause an incorrect driver to be loaded if Windows recognizes the vendor and product as one of its supported devices. Because there aren't any custom Windows drivers for this device, Windows will load the default HID driver when it enumerates the device.

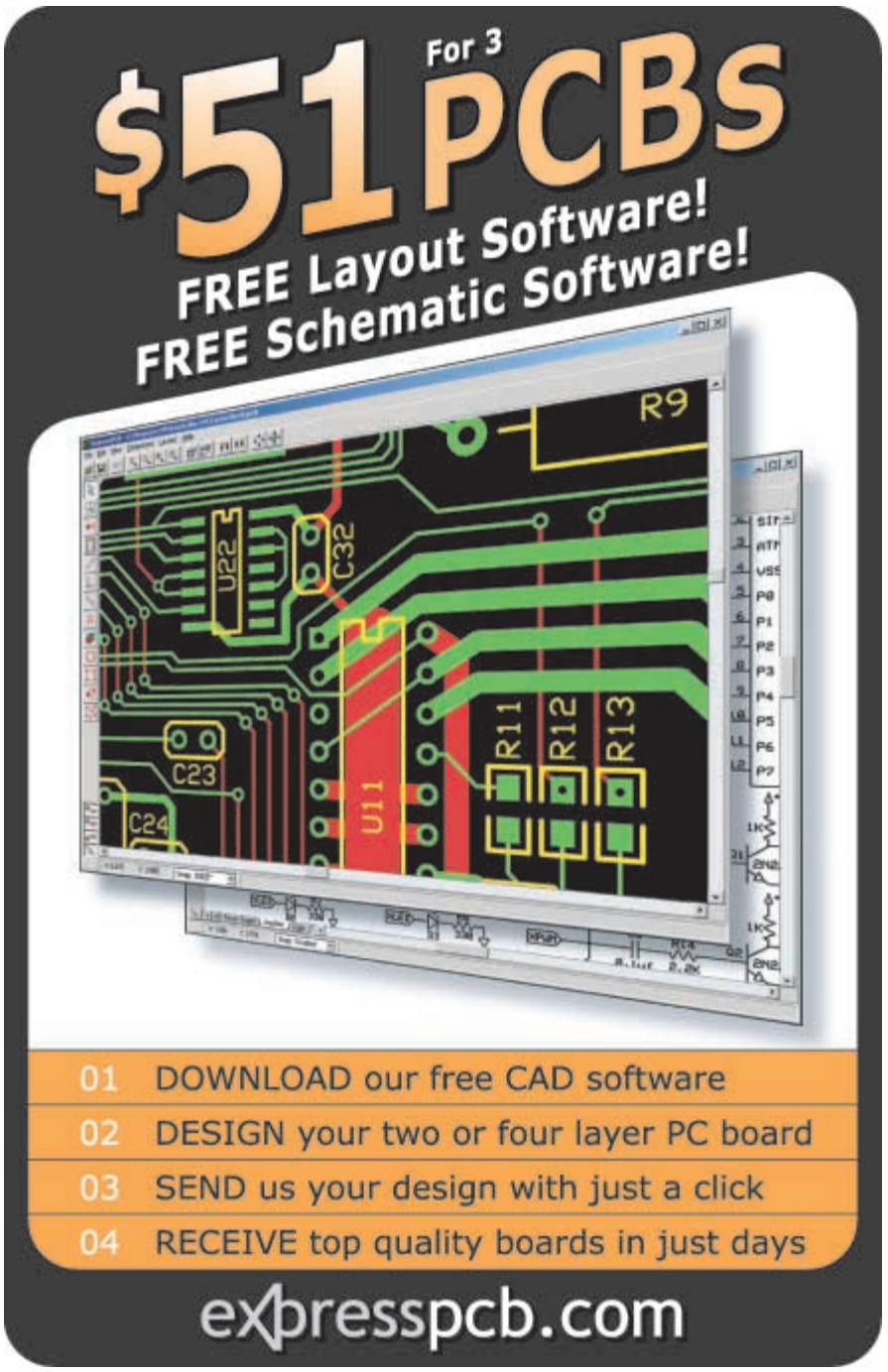
Because the HID is a general-purpose device that can support a mouse, keyboard, or just about any other low-speed device you'd like to build, it requires a special descriptor to define the data that will be transmitted to and from the device. The format of the report descriptor is rather complex, so the USB Implementers Forum developed the Descriptor Tool program (DT), which can create, edit, and validate HID report descriptors. DT was used to prepare the report descriptor for the Irritating Mouse test. The report descriptor also specifies that the x and y movements are relative to the previous position.

The `Process_Req` routine is the program's workhorse that handles all of the control requests from the PC. It first checks the BDnSTAT of the active endpoint to see if there is a token in, token out, or token setup request. If it's a token in, the program will continue sending the requested descriptor. If it's a token out, the program will continue processing it. If it's a token setup, the program will look at the contents of the buffer to determine the bmRequestType and bRequestType. For example, if the bmRequestType is 80h and bRequestType is 6, the program will look at the wValue of the buffer (1), which means this is a Get Device Descriptor request. The program will then begin to load the device descriptor into the EP0 input buffer. And so the processing continues.

After the PC has requested and received all the needed USB descriptors, it sets the configuration for the device, and mouse data is loaded into EP1 input buffer. The mouse data consists of 3 bytes of data: the mouse button data (unused), the x position, and the y position as defined in the HID report descriptor. The data is updated in 100-ms intervals, and the mouse begins to move in figure eights (extremely irritating).

DEBUG THE PROGRAM

Initially, I tried debugging the software using the five LEDs on the cir-



Debug Output	Comment
H>D	Host to device
SC	Set configuration
01 C	Configuration is set to 1
00 ff 00	Mouse data is transmitted from EP1 once
HID	HID request BufferCopy.bmRequestType & 0x20
H>HID	Host to HID
SIdle	Set idle
stall	Stop EP1 transmissions
I>H	Interface to host
HID RD	Report descriptor
05 01 08	50 bytes of report descriptor are sent
a1 00 08	
15 00 08	
81 02 08	
05 01 08	
7c 5a 08	
f5 0a 02	
H>D	Host to device
SC	Set configuration
00 A	Configuration is set to 0, which forces device back into address state and prevents mouse data from being sent.

Table 3—*In this debug output from a run with a compiler error, the HID report descriptor was transmitted incorrectly, which caused the host to take the device out of Configured mode.*

cuit board, but that provided inadequate information. Because the compiler prevented me from using the same subroutine in two different threads, I used LEDs to report information about the interrupt routine. I hooked up an optional Maxim 233 RS-232 driver/receiver to the microprocessor UART to send debugging data via a serial port to the HyperTerminal communication program running on the PC. The first LED lights up when there's a reset interrupt. The fifth LED lights up when there's an error interrupt.

Table 3 is part of the commented debugging output from one of the early runs that revealed an error. In this run a compiler error caused the report descriptor to be stored incorrectly and the USB enumeration failed. The problem occurred in the fifth buffer transfer of the HID report descriptor. The first 2 bytes of the sixth buffer were transmitted as 7C5Ah instead of the correct 257Fh.

The faulty report descriptor transmission caused the PC to change the configuration from 1 back to 0. With the latter, the device is in the addressed state and no mouse data transmissions can occur. Table 4 shows the commented output of a run after the compiler error was fixed. To turn off the debug output, the `#define DEBUG_SERIAL 1` line in USBDEFS.h should be commented out.

SIMPLE USB

I chose a chip and software development system and built a circuit board. I also tested the circuit board with a simple HID mouse application. Now it's your turn.

The PIC18F2455 provides an excellent platform for building a wide range of USB devices. The BOOSTC C compiler is a low-cost alternative to more expensive compilers for PIC software development. Good luck.

Robert Lang is a professional electrical engineer interested in embedded microprocessors, mechanical musical instruments, and MIDI. He has written several articles for electronics, computer, and synthesizer magazines. You may contact Robert at rlang@netdoor.com.

PROJECT FILES

To download the code and a complete parts list, go to ftp.circuitcellar.com/pub/Circuit_Cellar/2005/178.

RESOURCES

E. Ackermann, *The Essentials of the C Programming Language*, Research & Educational Association, Piscataway, NJ, 2001.

Microchip Technology, "PIC18F2XX0/2XX5/4XX0/4XX5 Flash Microcontroller Programming Specification," DS39622D, 2005.

USB Implementer's Forum, "Device Class Definition for HID Version 1.11," www.usb.org/developers/hidpage/.

——, "HID Descriptor Tool," www.usb.org/developers/hidpage/Microsoft.

——, "Universal Serial Bus Revision 2.0 Specification," www.usb.org/developers/docs/.

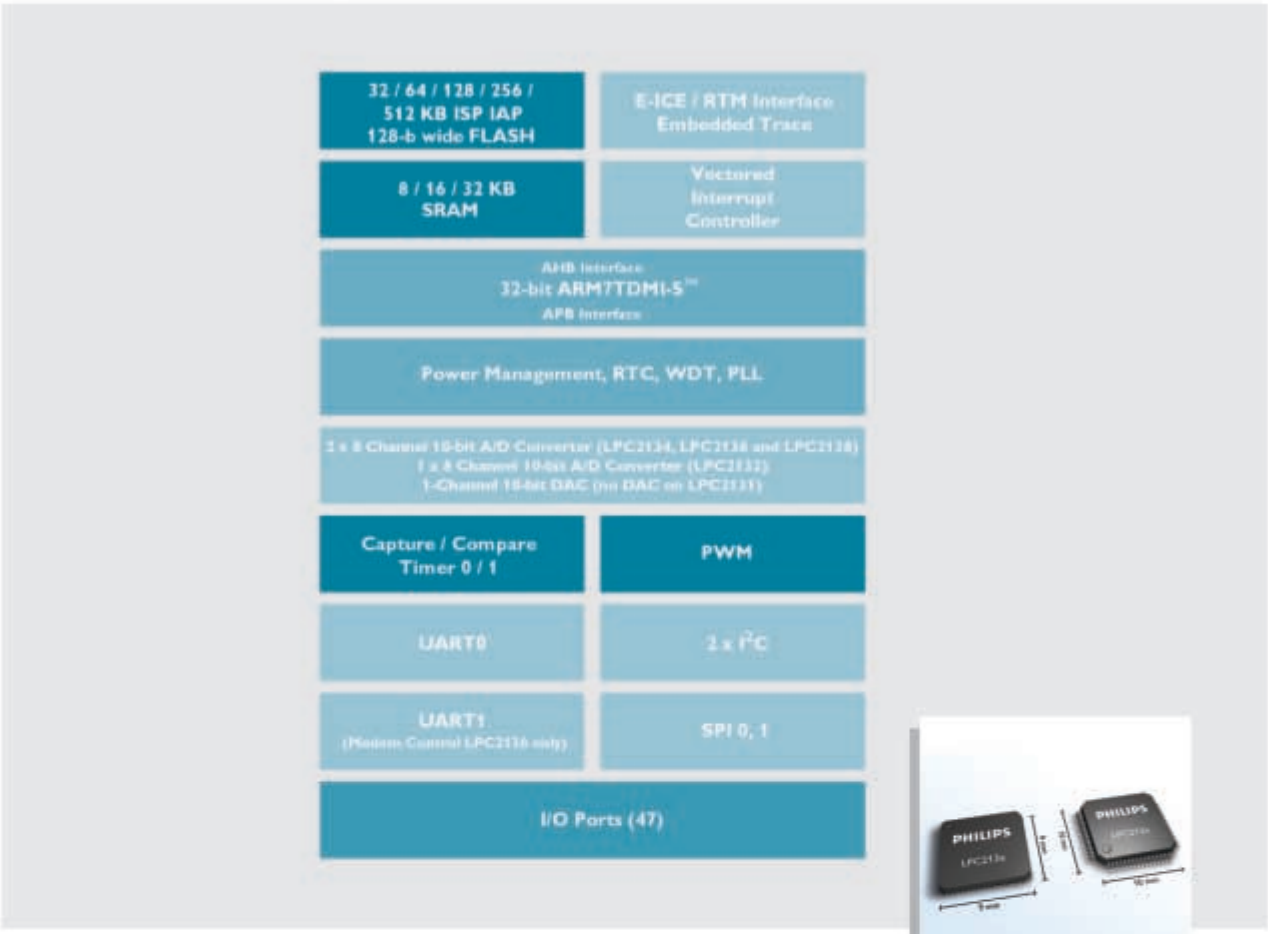
Debug Output	Comments
USB Irritating Mouse V1.1	Program debug begins
I	USB initialization complete
TMR0 up	Timer is working
D>H	Device to host
GD	Get descriptor
device	Descriptor type=device
12 01 08	Show first 2 bytes and total number of bytes transmitted in hex (8)
H>D	Host to device
SA	Set address
2	Address is 2
D>H	Device to host
GD	Get descriptor
device	Descriptor type=device
12 01 08	Get all 18 bytes of
d8 04 08	Device descriptor
00 01 02	
D>H	Continue getting
GD	Config and String
Config ...	Descriptors
H>D	Host to device
SC	Set configuration
01 C	Config is set to 1
00 ff 00	Mouse data transmitted once
HID	
H>HID	Host to HID
SIdle	Set idle
stall	
I>H	Interface to Host
HID RD	Report descriptor
05 01 08	50 bytes of report
a1 00 08	Descriptor are
15 00 08	transmitted
81 02 08	
05 01 08	
25 7f 08	
c0 c0 02	
00 ff 00	Mouse data transmission starts
...	Repeat eight times
00 ff 00	
00 ff 01	
...	Repeat eight
00 ff 01	Continues drawing figure
00 01 01	8 until unplugged

Table 4—*The HID report descriptor was transmitted correctly in this debug output from a run made after the compiler was fixed in version 1.9.3. The mouse data transmission starts after the host receives the good report descriptor.*

SOURCES

USBView
FTDI
www.ftdichip.com

PIC18F2455 Microcontroller
Microchip Technology
www.microchip.com



We've squeezed more in... how much can you squeeze out?

The gauntlet has been laid down. Take part in the **Philips ARM Design Contest 2005** and pit your design skills against your peers. Not only will you have the opportunity to win a share of $15,000 in prizes, but you can bask in the knowledge that your design pushes the industry's fastest ARM7 family of microcontrollers to its limit.

First Prize $3000
Second Prize $2000
Third prize $1000
Honorable Mention (8 prizes) $500 each

Plus five $1000 Bonus prizes:
$1000 bonus for the best project incorporating the 2138
$1000 bonus for the best project incorporating the 2136
$1000 bonus for the best project incorporating the 2134
$1000 bonus for the best project incorporating the 2132
$1000 bonus for the best project incorporating the 2131

Deadline for entries: October 24th, 2005.

Just hit **www.circuitcellar.com/philips2005** for the competition rules and entry form.

PHILIPS
sense and simplicity

© Koninklijke Philips Electronics N.V. 2005. All rights reserved.

AVR 2004 DESIGN CONTEST

Contest Winner

'Net Radio

Build an Internet Radio Receiver

Bertrand's ATmega64L-based 'Net Radio enables you to listen to Internet radio broadcasts without a PC. The 'Net Radio handles the Internet protocols and MP3 decoding tasks.

Most people enjoy listening to the radio. Some people use it for ambient background music. Others like listening to debates and talk shows. What about you? It probably depends on the types of programs offered in your area. Some regions are covered by dozens of stations, while others aren't covered at all. Times are changing, though. Many radio stations are now broadcasting programs over the Internet, and if you have a relatively fast Internet connection, you can use your PC to listen in. But there are drawbacks to this method. For instance, it wastes a lot of potential computing power. I built the 'Net Radio to address these problems (see Photo 1).

The 'Net Radio consumes a lot less power and takes up a lot less space than a PC. Think about it: you no longer have to worry about a keyboard, a mouse, a monitor, and wires cluttering up the room in which you currently keep your radio receiver.

'NET RADIO BASICS

Do you have a cable connection, a DSL connection, or a satellite connection? The protocols for each connection vary, from the IP packet's encapsulation (e.g., PPPoE and PPPoA) to the protocols used for authentication (from PAP to 802.1x). The 'Net Radio is compatible with different Internet connections. You can connect it to any Ethernet network. A router enables you to reach the Internet. This setup is becoming increasingly common because more and more homeowners are equipping their houses with LANs.

After I knew what I would interface the 'Net Radio with, I had to find a way to communicate with the network. I knew I would use a microcontroller for the project, but I didn't know how to connect it to an Ethernet network. Some microcontrollers have an embedded Ethernet interface, but they always need an Ethernet connector and additional components (e.g., magnetics and passive components) for a connection, and some of these components are hard to find in low quantities. Furthermore, microcontrollers are usually powerful (some have a CAN bus), and development tools for working with high-level languages (higher than assembler) are usually expensive. And I still had to build a programmer. That's why I decided to use a microcontroller without an embedded Ethernet interface: the Atmel ATmega64L.

I also used the extremely powerful and optimized AVR-GCC cross compiler, which is part of the WinAVR package. The GCC is the GNU C compiler used by designers around the world to compile huge, well-known projects (e.g., the Linux Kernel). And, thanks to the open-source community, it's debugged and free. I got an AVRISP through the Atmel AVR 2004 Design Contest, so I didn't have to worry about any programming issues. An AVRISP is an official Atmel programmer for the AVR family. One end connects to a PC's serial port. The AVR is connected on the other end via a programming connector.

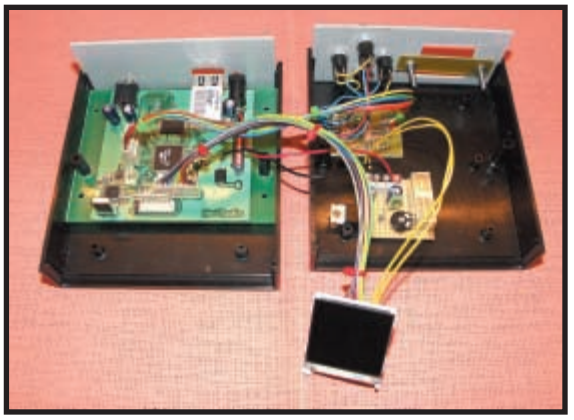


Photo 1—*No PC in sight! The main board is on the left. The LCD backlight power supply and switch debouncer board are on the right. The LCD is at the bottom.*

I used a Lantronix XPort to interface the microcontroller to an Ethernet network. The XPort is basically an Ethernet-to-serial converter. It has all the components for interfacing with an Ethernet network, including a microprocessor that handles all of the common protocol stacks (TCP/IP). This is important because the TCP/IP protocol is difficult to handle (because of fragmentation, for example) and writing (and debugging) a TCP/IP stack takes a lot of time. Furthermore, network radio is broadcast over the TCP/IP protocol. (Actually, it's broadcast over the HTTP protocol, which is encapsulated in the TCP/IP protocol.)

TCP/IP stacks aren't free. (If they were, you might run into code licensing problems.) Fortunately, the XPort solves this problem: it's royalty free. It also speeds things up and doesn't take up much more space than an average RJ-45 connector.

The user interface enables you to choose what you want to listen to, set the volume, start and stop playing, and so on. I thought it

would be fun and interesting (technically speaking) to use a graphic color LCD instead of a plain HD44780-like character matrix LCD. My color LCD is designed for Nokia mobile phones. It has 131 × 131 pixel resolution, and can handle 4,096 true colors. (It can handle 65,000 colors with dithering, but dithering is cheating, because the internal RAM can store only 12 bits per pixel.) The LCD has an SPI-like interface, which makes it easy to connect to the ATmega64L microcontroller. Last but not least, the module has a Philips PCF8833 integrated controller, which has enough RAM to store an entire frame. So, I just have to send to the screen what I want to update on the display. It's extremely easy to do and doesn't require a lot of computing power.

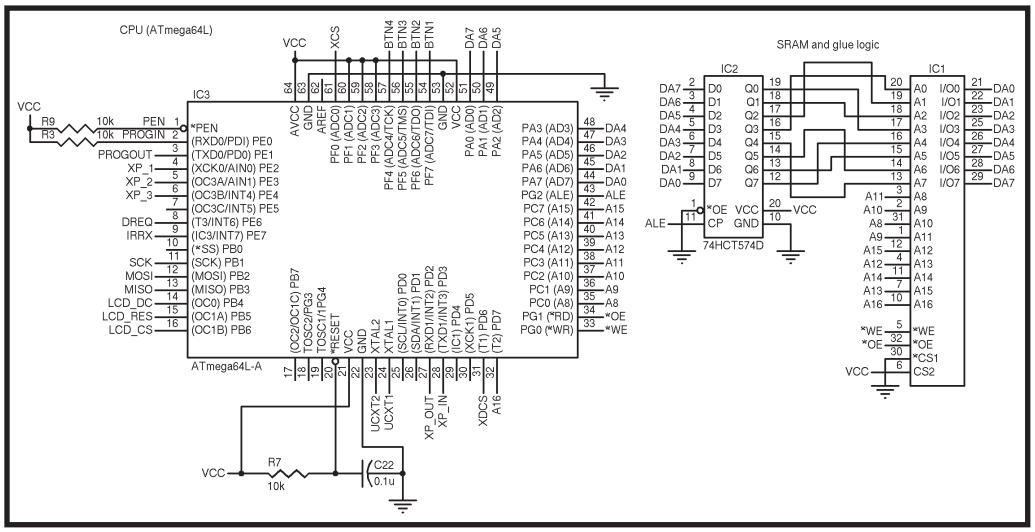


Figure 1—*The ATmega64L microcontroller is connected to various devices; in fact, nearly all of its numerous I/O ports and integrated devices are used in this project. You can clearly see the external SRAM connection through some glue logic (a latch) here. The external SRAM allows the 'Net Radio to buffer enough MP3 data for glitch-free playback.*

Fortunately, Internet radio broadcasts are sent as compressed audio. (Some lucky people have 100-Mbps Internet connections, but not many.) The most common format is the well-known MPEG I layer III (MP3). Of course, the ATmega64L microcontroller isn't powerful enough to perform the decompression. (It's a microcontroller, not a DSP.) That's why I had to find a suitable dedicated MP3 decoder chip. I used VLSI Solution's new VS1002, which performs all of the decompression and converts the uncompressed numeric audio signal to an analog audio signal through an on-board DAC.

Now that you're familiar with the project, let's take a closer look at the parts I used. I'll describe the software after I cover the hardware.

ATmega64L MCU

The ATmega64L microcontroller has 64 KB of flash memory (see Figure 1). The current code requires far less code space, but having a lot of code space made testing easy. For example, I had some problems initializing the VS1002. I encoded a small, 5-s, 32-kbps MP3 file to an .h file, so the MP3 data was stored in the ATmega64L microcontroller's flash memory. I created a small procedure that moved sequentially through different initialization settings, displayed the current configuration on the LCD, and tried to play the small MP3 file. When I heard the correct song, I looked at the LCD and wrote down the current configuration's ID.

I was also able to store the 256-color, boot-up logo in the flash memory. The logo is 17 KB long. It takes a little more than one-quarter of the flash memory!

The ATmega64L microcontroller's internal SRAM is only 4 KB. That's enough to store the code variables that control the program, but it isn't enough for buffering. Because the data comes from the Internet, I have to take care of a small buffer to prevent any skip.

I hooked up an external SRAM to the ATmega64L microcontroller (see Figure 1). The MCU has an XMEM interface that can drive an external SRAM as if it were part of the internal RAM (it's on the same address space), so the code doesn't have to take care of the external SRAM control signals and wait states. But the problem with the interface is that it's in the same address space as the internal 4-KB SRAM, so you lose 4 KB in the external memory. And because the AVR architecture can handle only a 16-bit address space, it would have limited the directly addressable memory to 64 KB, which is too small for this project.

Furthermore, I wanted to use the 128-KB memory chip I had on hand. The process of pagination (driving one of the address bytes in software) would have been hard because I didn't know how the compiler uses the RAM.

Even though I had hooked up the SRAM so I could still use the XMEM interface, I decided to drive the SRAM with a software driver. I didn't lose any space, but I used up some processing power. I then had 132 KB of RAM for my project, which was enough to buffer the MP3 data. A 128-kbps stream is 16 KBps, so I can buffer 4 s of music with the 128-KB buffer that I keep half filled. And I didn't have to worry about how the compiler uses the internal SRAM. The compiler takes care of it, and I take care of the external memory (divide to lead).

You must carefully program a high-end AVR microcontroller. In many cases, programming involves the same pins as those for the SPI interface. With an ATmega64 microcontroller, however, the program data in (PDI) and program data out (PDO) pins are separate. An external resistor pulls up the program enable (PEN) line. When I want to enter Programming mode, I ground the PEN line (the programming connector does this) and reboot the 'Net Radio. The AVR then waits for programming instructions.

There is one nasty trap with AVR devices: they use the external clock source for the programming sequence. If it's set incorrectly during the previous programming cycle, the device

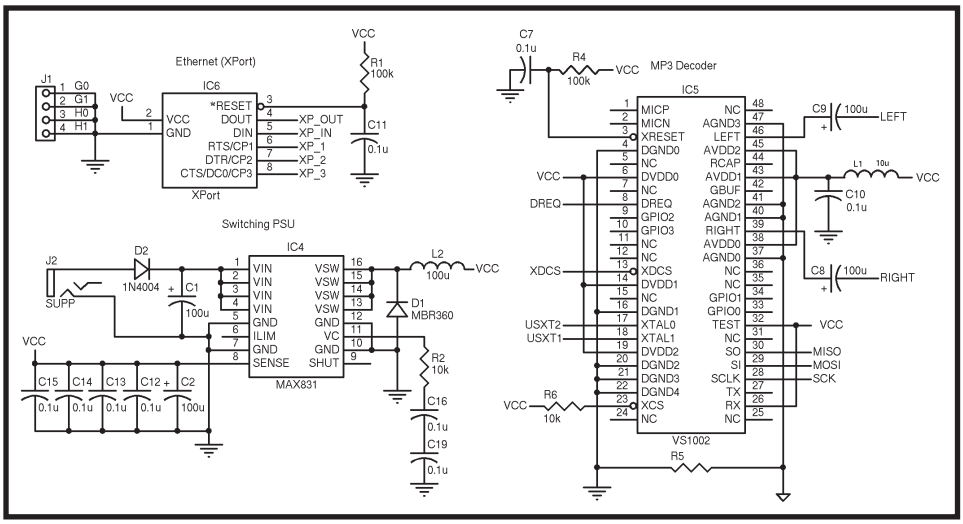


Figure 2—*The heart of the 'Net Radio is a microcontroller, but it cannot work alone. It is aided by several circuits. The XPort (top left) enables communication with the Ethernet network. The switching power supply (bottom left) generates the stabilized 3.3 V required to power the project. And last but not least, the VS1002 (right) handles the MP3 decoding tasks.*

will not boot. I made this mistake once, and I had to connect a signal generator to the clock pins to simulate an external clock. This problem doesn't exist with Microchip Technology's PIC microcontrollers, which don't need an external clock source when programming (apart from the data clock because the communication is synchronous).

Oh, I didn't tell you why I used the 'L'' version of the ATmega64. The 'L'' stands for low power. I didn't really need low power consumption. However, I needed low voltage because of the XPort, which can work only at 3.3 V, a voltage that can be supplied by a 120- or 240-V outlet through a transformer or, preferably, a switching power supply. But I needed a microcontroller that could work at 3.3 V, which is the only voltage at which the XPort can work. I found that running the system at 3.3 V was cleaner than running the microcontroller at 5 V and using a level converter to talk with the rest of the components.

ETHERNET CONNECTION: XPort

The XPort has seven pins: two for the power supply, two for the serial UART (TX/RX), and three GPIO pins that can be configured for flow control. The XPort is connected to the ATmega64L microcontroller through the second serial port and through three GPIO pins (see Figure 2).

You can configure the XPort through the Ethernet network with a Lantronix utility that enables you to set the serial port's settings (e.g., data rate and parity) along with the function of the three GPIO pins. The maximum data rate that my version supports is 230 kbps, which is enough to receive a 224-kbps MP3 stream. Note that 224 kbps is a standard MP3 bit rate. The highest bit rate I've seen for radios is 160 kbps. I think it's enough. The 'Net Radio doesn't need the faster version of the XPort.

MP3 DECODING: VS1002

The VS1002 MP3 decoder chip is connected to the ATmega64L microcontroller through the SPI interface (see Figure 2). An SPI interface is a synchronous communication bus. The master (the MCU) provides the clock. There are two data lines, one coming from the slaves (e.g., the VS1002) and one coming from the master. There are also chip select (CS) lines for the slaves. When a CS line is active, the slaves listen for data on the bus and eventually respond to it.

The VS1002 has two CS lines, one for a control bus (XCS) and another for a data bus (XDCS). The former enables it to send control commands to the VS1002 (initialization, digital volume control, and so on). The latter enables it to send and receive MP3 data. The 'Net Radio is used only as an MP3 decoder, so only MP3 data is sent to the VS1002.

The VS1002 operates in Normal mode and Streaming mode, which I found especially useful. I wasn't able to find another coder/decoder that I could configure to work in Streaming mode.

In Normal mode, which is common to most if not all MP3 codec chips, there's a DREQ signal that the decoder sets to 1 when it wants data. The only thing to do is to read the MP3 file (from a flash memory card or a hard drive) and send it to the decoder when the line is a 1. When it goes to 0, you must stop sending data to the decoder and wait for it to go to 1 again. Then you can resend data.

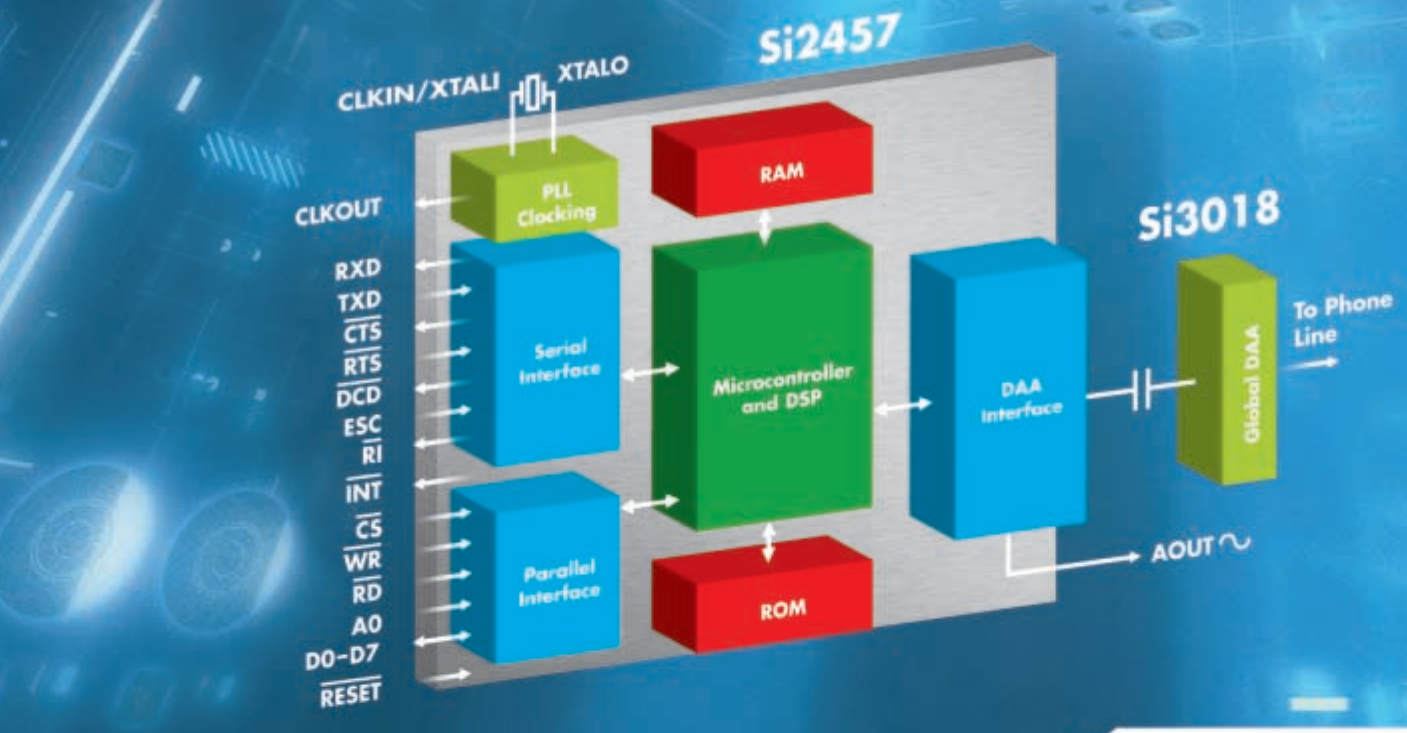
There's a problem with Normal mode that makes it inappropriate for the 'Net Radio. In Normal mode, the playback rate is controlled by the MP3 decoder's clock. Perhaps a 128-kbps MP3 will be played at 129 kbps instead of the nominal 128 kbps because of a clock drift. It isn't a problem when you can stop reading the data and restart whenever you want. But when a remote server sends the data, you can't stop reading, even if it sends data too fast. In fact, if you do, you'll lose data and it will result in sound glitches.

In Streaming mode, you continuously send MP3 data to the decoder. Nothing guarantees that you'll receive the packets in the same order that they have been sent. This is especially true when you're working with the Internet. Thus, the XPort must reorder the packets before you can send them to the ATmega64L microcontroller, which can result in a short or random delay. Furthermore, the Internet doesn't guarantee that the delay between each packet always will be the same. This results in another random delay.

So, if you barely read the data from the network stream and send it to the decoder, you'll have problems. If the server's clock has a small drift compared to the decoder's clock, your buffer will either slowly fill itself or run out of data.

The VS1002 family's special Streaming mode addresses all of the aforementioned problems. It automatically adjusts its data rate to keep the buffer 50% full. If the buffer is more than 50% full, the VS1002 will

One modem design for all products. All speeds. All countries.



ISOmodem®

The ISOmodem family of products are pin-compatible complete two-chip modems that support all speeds (2400 bps to 56 kbps) with a single PCB layout. Utilizing Silicon Laboratories' integrated silicon DAA technology, the ISOmodem offers a programmable isolation interface to meet global telephone line requirements. The ISOmodem is an ideal solution for embedded modem applications where a simple host interface, small footprint, low power consumption and global compliance are required.

Highest level of integration	75% less board space with integrated DSP data pump, controller, codec and 2-4 wire hybrid, eliminating the need for relays, opto-isolators and isolation transformer
Simplicity	Quick and easy-to-use due to serial UART, AT commands and an integrated global DAA
Global compliance	Fully programmable to meet global standards with a single design

Key benefits

- Single layout for all speeds
- Small footprint
- Low component count
- Globally compliant
- Low power consumption

Applications

- Set-top boxes
- Digital video recorders
- Point-of-sale terminals
- Security systems
- Industrial/medical monitoring devices
- Internet access devices

For more information, visit www.silabs.com/ISOmodem

WIRELINE | WIRELESS | OPTICAL | MICROCONTROLLERS

SILICON LABORATORIES®

www.silabs.com

©2004 Silicon Laboratories Inc. Silicon Laboratories and the Silicon Laboratories logo are trademarks of Silicon Laboratories.

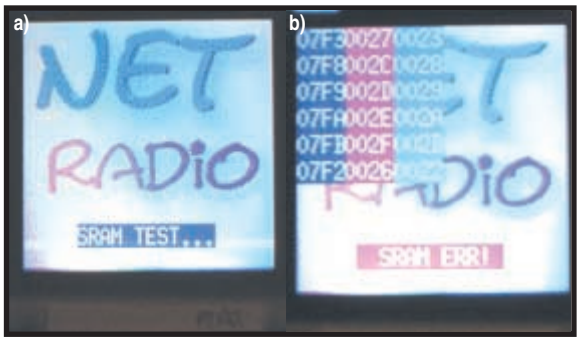


Photo 2a—*When the 'Net Radio boots, it checks the integrity of its external SRAM to be sure that the buffering will be done correctly.* ***b***—*If there's a problem with the external SRAM, the failing addresses are displayed along with what's actually been read and what should have been read.*

increase the data rate until the buffer returns to 50% full. At that point the VS1002 will return to a nominal data rate. If the buffer becomes a little too empty, it will slow the data rate until the buffer is 50% full again.

The VS1002 decoder was perfect for my 'Net Radio project, although it has some features that I didn't use. It can encode data in ADPCM (WAV) and read VLSI's MP3+V format, which is a modified MP3 format with higher quality for the same bit rate. I could have used the encoding feature to convert the 'Net Radio into a bandwidth-hungry network radio station, but I didn't want to build a device for spying.

COLOR LCD

The LCD module consists of a controller, the LCD, and the backlight assembly. Like the VS1002, the controller is connected to the ATmega64L microcontroller through the same SPI interface. It has its own CS line ("LCD_DC" in the schematics posted on the *Circuit Cellar* ftp site).

The SPI interface of the LCD is special because there's only one clock and bidirectional data line. But it wasn't a problem because I didn't read any data back from the LCD. It was as if it had only one receiving line.

Another special thing about the LCD controller interface is that it works with 9-bit data packets. That was a problem because the AVR SPI interface can send a maximum of only 8 bits per packet. So, I used a small software trick to overcome this difficulty (see Listing 1).

The backlight of the LCD module consists of two serially connected white LEDs that illuminate a light diffuser under the LCD glass. The LEDs required approximately 6.5 V at 15 mA, so I quickly hooked up and then used an LM317 as a voltage generator. Used in this manner, an LM317 can generate from 1.25 to 35 V, provided its input voltage is high enough. I used it to generate 7.5 V. I used a resistor to limit the current.

As you can see in Figure 2, I used a switching power supply for the main 3.3 V. The power supply is based on a Maxim MAX831, which is a fixed-output, step-down converter.

SOFTWARE

The 'Net Radio's firmware is in the ATmega64L microcontroller. (You can also modify the XPort's firmware if necessary.) I used Programmers Notepad for a development editor. You may download the code from the *Circuit Cellar* ftp site.

The source code consists of approximately 20 files and roughly 400,000 characters (including the fonts, etc.). There are numerous files—in fact, there is one per hardware function (MP3 decoder, LCD, etc.). AVR-GCC can compile the files using the make file. Programmers Notepad can open the project with the netradio.pnproj project file. Warnings aren't generated during compilation, which is a good start.

At start-up, the firmware initializes all the peripherals and performs basic tests like an EEPROM checksum test and an SRAM test. If there's a problem, the firmware will inform you and even try to repair the data (e.g., by

Listing 1—*A little trick allows you to send a 9-bit word to the LCD controller through the 8-bit SPI. You can send the first bit by deactivating the 8-bit SPI. The remaining bits are sent with the SPI.*

```
void
spi_sync_transmit (unsigned char cData)
{
       SPDR = cData;
              while (!(SPSR  &= (1<<SPIF)));
              //Wait until the byte is sent.
}
void
lcd_command (unsigned char cData)
{
              cbi(SPCR,SPE);   //Disable SPI
              cbi(PORTB,PB2);  //Set MOSI low
              sbi(PORTB,PB1);  //Set SCK high
              cbi(PORTB,PB1);  //Set SCK low
              sbi(SPCR,SPE);   //Enable SPI

              spi_sync_transmit(cData);
}
void
lcd_data (unsigned char cData)
{
              cbi(SPCR,SPE);   //Disable SPI
              sbi(PORTB,PB2);  //Set MOSI high
              sbi(PORTB,PB1);  //Set SCK high
              cbi(PORTB,PB1);  //Set SCK low
              sbi(SPCR,SPE);   //Enable SPI

              spi_sync_transmit(cData);
}
```

Listing 2—*An array like this is difficult to handle if you want to save memory because it's always located in RAM. You have to use a special function if you want to read the data directly from the flash memory.*

```
const static unsigned char example_array[] = {
0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06,
};
```

restoring defaults in the EEPROM that are stored in the flash memory). Photo 2a shows the 'Net Radio performing an SRAM test. Photo 2b shows what happens if there's an SRAM failure. In the event of a failure, the system will display some of the failing addresses, what's read, and what has been expected. This enables you to easily diagnose the problem.

Because of the AVR architecture, you have to be careful when you're programming in C language. For example, when using an array, data is copied from the flash memory to RAM and then initialized, even if it doesn't have to be modified (see Listing 2). Because of this behavior, it isn't possible to store large arrays (e.g., fonts and boot-up logos) directly. The trick is to tell the compiler to store the data in flash memory and not copy it into RAM. Then you can use special functions to read the data directly from the flash memory. This frees up a lot of RAM. Refer to the lcd.c file on the *Circuit Cellar* ftp site for an example. The `lcd_puts_P` function puts a string stored in the flash memory on the LCD.

Now let's focus on the problem of the LCD controller working with 9-bit data packets instead of 8-bit data packets. The first bit is 0 for command and 1 for data, so I wrote two different procedures. The `lcd_command` command disables the SPI, sends a 0, and then sends the 8 bits of the command using the hardware SPI. The `lcd_data` command does the same thing, but it sends a 1 before the data instead of a 0.

The code in Listing 1 sends a 9-bit word to the LCD controller. Figure 3 is the transcript of a typical communication between the 'Net Radio and a network radio station. Wow, this stuff looks complex. Fortunately, the 'Net Radio is easy to use. Let's take a look at the user interface.

USER INTERFACE

As I was working on the 'Net Radio, I knew a problem would arise. Unlike conventional radios, the 'Net Radio needed a complex address to connect to ~~at~~ least an IP address made with four numbers (it will become more complicated when the Internet switches to the IPv6 protocol) ~~or~~ a host name. The latter is harder to enter because it must contain at least one letter, so you can't use a numbers-only keyboard. I didn't want to add a full keyboard to the 'Net Radio because I wanted to keep

Legend:
→ To the network radio station
← From the network radio station

→ GET */path of the stream HTTP/1.0*
→ Host: *hostname or ip of the server*
→ User-Agent: NetRadio/1.0
→ Accept: */ *
→ Connection: close
→
← ICY 200 OK
← icy-notice1: *some text from the server*
← icy-notice 2: *more text from the server*
← icy-name: *common name of the radio station*
← icy-genre: *genre of the radio station (e.g., classical or techno)*
← icy-url: *http://www.url of the radio.com*
← content-type: audio/mpeg
← icy-pub: *1 or 0 whether or not the stream is public*
← icy-br: *bit rate of the radio in kilobits per second*
←
← *<MP3 binary data start from here>*

Figure 3—*In a typical exchange between a network radio server and the 'Net Radio, the latter sends a request with some parameters. The server responds with a header followed by the MP3 data.*

Save months of development time by using our ready-to-run single board computers with your choice of operating system...
VIPER single board computer
• 400MHz Intel® PXA255 processor
• Ultra-low power 1.9W (typical)
• TFT/STN graphics controller
• 64Mbytes DRAM / 32Mbytes Flash
• Ethernet, USB and serial ports
• CompactFlash (CF+) and PC/104 bus
MERCURY single board computer
• 533MHz Intel® IXP425 communications processor
• x2 100baseTx Ethernet ports
• Accelerated encryption hardware
• x4 USB 2.0 and x4 serial ports
• 64Mbytes DRAM / 32Mbytes Flash
• CompactFlash (CF+) and PC/104 bus
Windows® CE .NET Development Kit
• Uses Microsoft Transaction-Safe FAT File System
• Comprehensive set of Windows CE components
• Includes Embedded Visual Tools C++ Tools
Embedded Linux Development Kit
• Based on 2.6 kernel
• Uses Journalling Flash File System (JFFS2)
• OpenSSH for improved communication security
VxWorks® Development Kit
• Support for VxWorks 5.5 real time OS
• WindML 3.0 multi-media library
• Automatic Tornado II installations
Now bundled with FREE CF WiFi card!*
* not included in VxWorks Development Kit
Compact enclosure with 320 x 240 color TFT display and analog touchscreen.
Combining the very best of low power Intel XScale® RISC technology & the industry standard PC/104 format.
Arcom
888-941-2224
www.arcom.com
Think Embedded. Think Arcom.

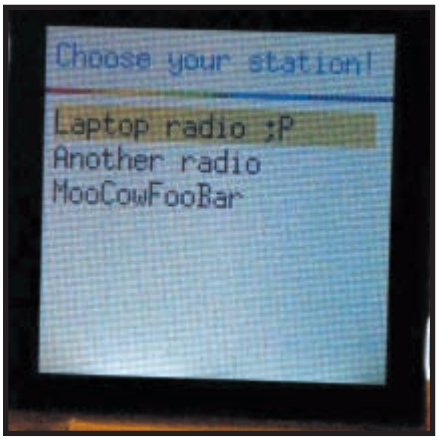


Photo 3—*After the 'Net Radio has booted up and completed its self-tests, it displays a selection menu of accessible radio stations. You can connect to one of them with a simple press of a button.*

the system as small as possible. (One of its best features is its small size!) I also wanted a commercial-quality system that I could eventually sell without having to make a lot of changes.

I stored the network radio station list along with other data in the ATmega64L's data EEPROM. Ideally, however, I would have created a central server on the Internet to store a list of network radio stations. Each 'Net Radio would have a unique identifier stored somewhere in its flash memory, and the flash memory would be protected from reading and writing. After booting up, the 'Net Radio would ask the central server to send it the station list based on its unique ID. All of this data then would be stored on the EEPROM, and you'd have the choice of listening to any one of several stations.

Under these ideal conditions, the list size would be limited only by the size of the EEPROM. I would be able to add an external EEPROM or even a 1-Gb memory card for additional space. In addition, it would be extremely easy to give you access to both free and subscription-based radio stations while letting you configure your 'Net Radio on an easy-to-use interface. With such a system, you could easily add fancy features to your system and modify the screen's colors.

I couldn't build my ideal system because I didn't have a server on the 'Net that could host my central server. However, the 'Net Radio would work with such a system if you were to add a few lines of code that would allow it to retrieve its configuration from a 'Net server (a kind of bootloader).

You're already ahead of the game because the 'Net Radio can read the station list from its EEPROM. Plus, I've already built a tool for creating EEPROM images from a list of radio stations and configuration settings. This allowed me to keep the user interface simple. You can select radio stations and control the volume. Even people who don't like computers and cell phones can use the 'Net Radio. It's extremely easy to operate.

STATION IDENTIFICATION

After your 'Net Radio identifies the stations it can connect to, it will present you with a station list. Use the Up and Down keys to scroll through the list (see Photo 3). After highlighting the station you want, press the Right key. This will connect you to the station by sending the correct initialization sequence to the server. The system will then receive and play the MP3 stream. The main menu will reappear on the screen if the connection fails.

You can stop playing an MP3 stream at any time. Press any key other than the volume controls to return to the main menu.

STAY TUNED

I think you'll like the 'Net Radio. I really enjoyed building it.

I'm actually thinking about designing a second version that would be easier and cheaper to build. I also want to rewrite the firmware to include more features. Imagine using the 'Net Radio as an MP3 streaming server.

Stay tuned. In the meantime, start surfing the 'Net for radio stations that you like. This way you'll be ready to go when you finally have your own 'Net Radio!

Author's note: I would like to thank my friend Christophe Riehl for his help with this project.

Bertrand Archard is an engineering student at the Ecole Supérieure d'Ingénieurs en Electronique et Electrotechnique (ESIEE) near Paris. When he isn't studying, Bertrand enjoys working on microcontroller-based electronics projects. You may contact him at ba@cykian.net.

PROJECT FILES

To download the code and additional schematics, go to ftp.circuitcellar.com/pub/Circuit_Cellar/2005/178.

RESOURCES

AMIC Technology Corp., "LP62S1024B-T Series: 128K × 8 Bit Low Voltage CMOS SRAM," v. 1.2, 2004, www.amictechnology.com/pdf/LP62S1024B-T.pdf.

Atmel Corp., "Atmega64/64L: 8-bit AVR Microntroller with 64K Bytes In-System Programmable Flash," 24901, 2001, www.atmel.com/dyn/resources/prod_documents/doc2490.pdf.

AVR libc information, www.nongnu.org/avr-libc/user-manual.

J. Bachiochi, "Global XPortation," *Circuit Cellar*, 162, January 2004.

Lantronix, "Xport User Guide," 900-270, rev. E, 2004, www.lantronix.com/pdf/XPort_UG.pdf.

Philips Semiconductors, "PCF8833: STN RGB 132 × 132 × 3 Driver," 9397 750 10059, 2003, www.semiconductors.philips.com/acrobat_download/datasheets/PCF8833_1.pdf.

VLSI Solution, "VS1002d MP3 Audio Codec," v. 0.71, 2004, www.vlsi.fi/datasheets/vs1002.pdf.

——, "VS10XX Application Notes," v. 0.72, 2004, www.vlsi.fi/datasheets/vs10XXan.pdf.

SOURCES

LP62S1024B SRAM
AMIC Technology Corp.
www.amictechnology.com

ATmega64L Microcontroller
Atmel Corp.
www.atmel.com

XPort
Lantronix, Inc.
www.lantronix.com

PCF8833 Driver
Philips Semiconductors
www.semiconductors.philips.com

VS1002 MP3 Audio decoder IC
VLSI Solution
www.vlsi.fi

Need Device Connectivity To The Pervasive Internet?*

Save Money, Save Time , Use The PowerCore

"Based on experience, I can tell you that Rabbit stuff works before you can get it all out of the box."

–Fred Eady, *Circuit Cellar*

The Core of your Embedded System

The PowerCore FLEX is a complete Rabbit microprocessor system with optional features such as a power supply, Ethernet, and a low-cost rugged A/D system. It plugs into a motherboard you design. You pay only for the options you need, and we quickly manufacture it. Development Kits include a powerful software development platform and extensive libraries.

From $29 qty. 5000 **Pay Only For What You Need**

Network and Internet Support

Wi-Fi

Your embedded device can network and serve web pages. Hardware and software supports Ethernet, Wi-Fi, and cellular telephone data networks. TCP/IP and most associated protocols are included in the Development Kit. SSL secure server software and other premium software modules are available at a nominal cost. With the implementation of optional software modules such as RabbitWeb, SSL, PPP, and AES, you can securely and conveniently communicate with your embedded device from anywhere in the world.

Configurable Features

- Microprocessor Speed
- On-Board Power Supply
- SRAM / Flash
- Serial Data Flash
- Ethernet Option
- A/D Converter System
- Battery-Backed SRAM

Build Your Own

Standard Features

- 39 General Purpose I/O
- 5 Serial Ports
- Real-Time Clock
- Watchdog / Supervisor

Development Kit Includes

- A loaded PowerCore FLEX with all options or non-Ethernet low-end version
- Prototyping board with development area.
- Documentation on CD.
- Dynamic C integrated development environment.
- Serial cable, power supply, and more.

* *Pervasive Internet: Embedded devices providing internet connectivity for commercial and industrial applications. Learn more at* ***www.powercoreflex.com***

Buy A Development Kit – From $129!

Includes Dynamic C (IDE) and hardware development tools. For a limited time, get a free copy of "Embedded System Design Using the Rabbit 3000 Microprocessor" with purchase.

www.PowerCoreFLEX.com $49 Value

RABBIT Semiconductor

2932 Spafford Street, Davis, CA 95616
Tel **530.757.8400**

USB 2.0 Interface

Daniel's inexpensive USB interface features an ADC0820 A/D converter chip and an EZ-USB interface. The system can sample 100,000 bps and send 512 bytes in 1 ms to a laptop.

The laptop industry has been gradually moving away from serial and parallel ports in favor of Universal Serial Bus (USB). In anticipation of the day when USB is the only available port for building interfaces to measure the external world, I developed an interface using National Semiconductor's inexpensive ADC0820 ADC chip and an EZ-USB interface.

The system can sample 100,000 sps and send 512 bytes in 1 ms to a laptop using Visual C++ 6.0 to display the voltage samples or a Fast Fourier transform (FFT). The FFT appears 50 times per second, which is sufficient for real-time frequency analysis. Cypress's EZ-USB kit is expensive, but it was the only one of the four systems that I tried that provided enough guidance and examples to develop my own application for the USB 2.0.

In this article I'll show you how to develop your own interfaces with USB. I'll also help you avoid several pitfalls.

WHY USB 2.0?

For my first USB project, I used an ActiveWire USB board to connect an ADC0820 to a 16-bit interface. It worked well, but because it was a USB 1.0 device, only 1 byte per millisecond was sent to the laptop. It took 0.512 s to send 512 bytes. The EZ-USB 2.0 interface, however, can send 512 bytes per millisecond, which speeds up the process considerably. It uses the bulk transfer method, which sends 512 bytes as a block of data. Note that it isn't a continuously sampling system, but it works well for FFT displays.

You can use a D-Link USB2.0 port CardBus if you don't have a USB 2.0 system installed and you have a CardBus interface (for PCMCIA cards). The card, which comes with a driver, works well, but it sometimes causes the laptop to stall at start-up when the USB board isn't connected. You can shut down by disconnecting your laptop's AC power and battery. When you restart the computer, it will load the drivers successfully. Never panic.

EZ-USB

The secret is in the 8051 chip on the EZ-USB board. The EZ-USB 8051 architecture has numerous areas of memory called "endpoints" that you can use as storage buffers. Endpoint 8 is for data transfers. The key thing about endpoints is that the 8051 can fill them with the data it collects from your hardware. The USB software can access the endpoints and send the data to the laptop.

I built my 8051 program by modifying Cypress's example on the kit's CD-ROM. I then downloaded it via the control panel application, which was also on the CD-ROM.

The 8051 program takes the data from the ADC0820 and puts it into endpoint 8. When the endpoint 8 buffer is full, the EZ-USB board's USB software sends the data in the buffer to the laptop. The 8051 program then loops back to gather more data and fill the buffer.

The program in the laptop uses Visual C++. It incorporates a modified routine on the EZ-USB CD-ROM to request the data from the 8051.

When I tried to slow down the 8051 program to take fewer samples per second, the system appeared to suffer an interrupt. This occurred when the sample time was greater than about 28 ms. There's a way to get around this. Take the samples at a high rate and discard the unwanted samples. I chose a 20 ms sample time, which gave me 50,000 samples per second. That's 50 samples per millisecond, so it took roughly 10 ms to fill the 512-byte buffer.

The 8051 endpoints use double-buffering, which means you have to send the data to the endpoint twice, so the data is sent to the laptop approximately every 20 ms in a 1-ms time frame. This results in a display rate of 50 times per second.

The C++ display screen enables you to take a block of 512 data points and display data or step through each value using the index of the data array. The other mode is used for entering the number of cycles (each cycle is a block of 512 data points) and displaying the FFT. This particular application was set up to display frequencies up to 2.5 kHz, which is good for showing voice frequencies, guitar frequencies, and signals coming from a ham radio code filter. I created the scale on the display by feeding in known frequencies from an audio signal generator and literally writing in the appropriate scale. With a sampling frequency of 100 kHz, you could have a scale that goes to 50 kHz (the Nyquist limit).

APPROACH THE PROBLEM

If you're interested in USB, you've probably spent some time researching the subject. By reading books like Jan Axelson's *USB Complete* and John Hyde's *USB Design By Example*, I learned that I needed the bulk transfer data transfer method, which transfers a block of data (512 bytes in this case) from the 8051 endpoint to the laptop. C++ receives the data in an array.

The standard procedure is to get a known application running by way of programs from the EZ-USB CD-ROM. If the gods are kind to you, the source code will be there too. After you find a program similar to your application, you can begin modifying it. I focused on a program for the 8051 and a program for the laptop.

In typical C and C++ programs, there are several files that you must compile and link to make the executable file. You should use programs from the EZ-USB CD-ROM that have source code files. Typically, there's a subdirectory for each example that includes all the associated programs.

HACK THE CODE

Let's start with the program for the 8051. You can test your program with the Cypress control panel, which you can install with the EZ-USB CD-ROM. Mind you, this isn't the control panel furnished by Windows XP. The CD-ROM installs the control panel under the Cypress entry that appears when you click the "All Programs" bar in the Windows Start box.

The 8051 program is written in C and uses the Keil compiler on the EZ-USB CD-ROM. Install the compiler during the set-up process. After the program is compiled, download the executable file to the 8051 using the control panel. You can also download the program to an EEPROM on the board. Typical applications have two C files that are compiled with other files to make the 8051 executable file. Listing 1 shows the code added to the fw.c file to take data from the ADC0820 and store it in the buffer.

If you install the EZ-USB files, the original fw.c file for the application will be located in C:\Cypress\USB\examples\FX2\bulksrc. I copied this entire directory to the FX2APPS directory, where I modified the fw.c file. The bulksrc directory contains the files needed to form the executable file. The file became ADCDATA8 after approximately 30 iterations.

The EZ-USB kit also includes "EZ-USB FX2 CY7C68013 Technical Reference Manual"(Cypress, 2001), which explains how to use output enable port B (OEB), output enable port D (OED), and send data to port B (IOB). I used port B for control signals to and from the ADC0820. I used port D to get the data from the ADC0820.

The EZ-USB board is double-buffered, so you must send the ADC data (512 bytes) to endpoint 8 twice. I placed the code right after the TD poll command within the main program built into the fw.c program.

Figure 1 (p. 32) is the wiring diagram I used to connect the ADC0820 to the 8051. A 7805 voltage regulator supplies 5 V for the chip. The large-value electrolytics help remove noise from the power supply. You should also put 0.1-µF capacitors at all the 5-V inputs to the chip.

The EZ-USB kit comes with an expansion board that connects to the main board via headers J1 through J6. I used only J1 because it contains both ports. A 34-pin connector (often used to connect floppy and hard drives to a motherboard) connects with J1. Photo 1 (p. 32) shows the ADC circuit and its connections to the EZ-USB board.

Port B bits are used for the start pulse (B0), output enable (B1), and INT (B2) for the ADC0820. The start pulse initiates the conversion process. The INT pulse lets the 8051 know that the conversion is complete. The output enable pulse enables the ADC0820's tristate outputs. The data from the ADC0820 output pins is read from port D (D0 through D7) and stored in the `EP8FIFOBUF` array. The

Listing 1—*I used this C code for the 8051. The routine was inserted in the fw.c program to load data from the ADC0820 to the 8051 endpoint 8 buffer. The system is double-buffered, and the data must be entered twice.*

```
TD_Poll();          //This command is already in the main "while loop."
REVCTL = 0x03;              //The code starts here.
SYNCDELAY;                  //This is a time delay.
FIFORESET = 0x80;           //Reset buffer
SYNCDELAY;
FIFORESET = 0x08;
SYNCDELAY;
FIFORESET = 0x00;
SYNCDELAY;
EP8CFG = 0xE0;              //Enable endpoint 8
OED = 0x00;                 //Make all of port D as input bits
OEB = 0x03;         //Make two lowest bits of port B the output bits
                            //The remaining 6 bits are input bits
for (i=0;i<512;i++)         //Loop 512 times
{
IOB = 0x03;                 //B0 is active low
IOB = 0x03;
ReadyLoop:  JSTATUS = IOB; //Read port B
  while (JSTATUS & 0x04)    //Keep looping if bit B2 = 1.
  goto ReadyLoop;
IOB = 0x01;                 //Send ADC output enable command to bit B1.
JDATA = IOD;                //Read data from port D
EP8FIFOBUF[i] = JDATA;
L=1;                        //Delay loop to slow down sampling frequency.
  for (k=0;k<5;k++)
  {  L=L+1;
  }
}
SYNCDELAY;
EP8BCH = 0x02;              //Enter number of bytes in buffer (high byte).
SYNCDELAY;                  //Enter number of bytes in buffer (low byte).
EP8BCL = 0x00;
for (i=0;i<512;i++)         //Do it all again for second buffer.
  {
  IOB = 0x03;
IOB = 0x02;
  IOB = 0x03;
  ReadyLoop2:  JSTATUS = IOB;
  while (JSTATUS & 0x04)
  goto ReadyLoop2;
IOB = 0x01;
JDATA = IOD;
EP8FIFOBUF[i] = JDATA;

L=1;                        //Delay Loop to slow down sampling frequency
  for (k=0;k<5;k++)
  {  L=L+1;
  }
}
SYNCDELAY;
EP8BCH = 0x02;
SYNCDELAY;
EP8BCL = 0x00;              //Code ends here
```

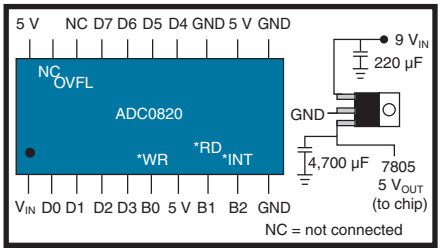


Figure 1—*The ADC connections run to the 8051's ports D and B. D0 through D7 are the ADC0820's data output pins. B0 starts the data conversion. B1 enables the data output pins. B2 tells the 8051 that the data is ready.*

routine loops to gather 512 data bytes. Finally, the buffer is declared full by putting the buffer count (512 bytes) in EP8BCH and EP8BCL.

When the buffer is declared full, the routine can't write to the buffer. The program simply skips over the `EP8FIFOBUFF[i]=JDATA` command and continues on. It then loops through the main `while` loop until the buffers become available again.

You can use the control panel to make sure the 8051 program works correctly. The numbers from the endpoint buffer represent actual voltage levels, so you can feed in different DC levels and test the ADC. The control panel also enables you to transfer data from the endpoint to the display screen by using the 'bulk int"button. Just make sure you replace the string length (64) with the new length (512). Also, you need to change the number of bytes to 512. When you're satisfied that the 8051 program is running correctly, it will be time to jump into the next phase, which involves developing the laptop program.

BACK AT THE LAPTOP...

The laptop program is based on the BulkXfer program located in C:\Cypress\USB\Examples\EzUsb\BulkXfer. Again, I copied the entire directory into a new directory (C:\adc_app). The C++ dialog file BulkXferDlg.cpp was modified to collect the data from the `EP8FIFOBUFF` array and store it in another array (`j_adcdata`). Listing 2 shows the routine that requests the data from the buffer and puts it into your array. After the data is in this array, you can use it for the various functions.

I discovered the correct pipe number by checking the 8051 software with the Cypress control panel. Using the Get Pipes button, I found that endpoint 8 was attached to pipe 3.

Photo 1—*The EZ-USB board is on the bottom. The expansion board fits over headers J1 through J6. The ADC board is fastened to the expansion board with 0.375″ spacers so you can easily remove it for your next project.*

I modified the BulkXfer dialog screen to include the functions I wanted for the FFT and data displays. Obviously, the BulkXferDlg.cpp file was extensively edited, which is why it isn't listed completely. All the files for the modified bulksrc and BulkXfer programs are posted on the *Circuit Cellar* web site.

Photo 2a shows a typical data display. Photo 2b shows an FFT display. The data

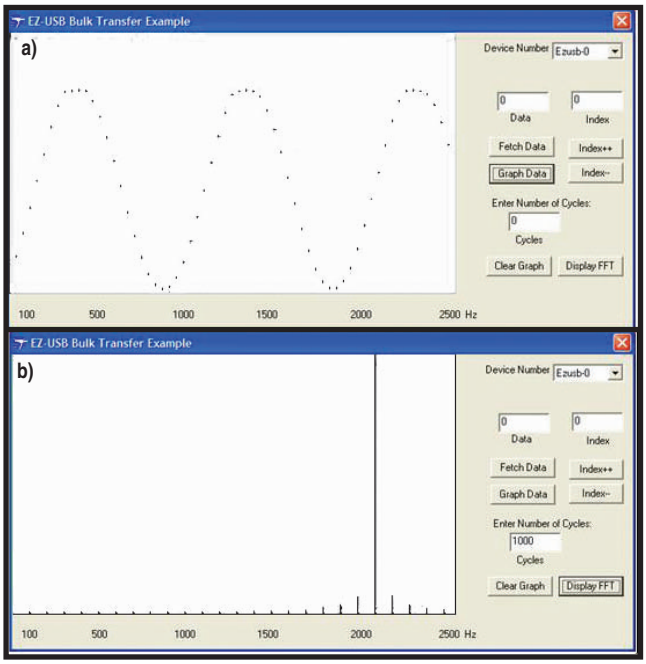


Photo 2a—*The display shows the first 64 bytes of a whistle sound input. Use the Index buttons to find additional bytes. The frequency scale is used only for the FFT display. The sample rate is 50 kHz.* **b**—*Take a look at one frame of the FFT routine. The system displays 50 frames per second, which enables you to display data in real time. The frequency scale shows the main components of the whistle, which is slightly lower in tone than the whistle in Photo 2a.*

comes from a single 512-byte block that's loaded after you click the Fetch Data button. The data is displayed after you click the Graph Data button. It's a good idea to click the Clear Graph button before taking the data because it'll give you a solid black background. Use the index buttons to inspect the data values one by one.

The FFT resolution was approximately 100 Hz, which was adequate for my work. I took the FFT routine, which uses the BASIC language, from Hal Chamberlin's book, *Musical Applications of Microprocessors*. I converted it to C++ for this application. The screen is updated 50 times per second, so I get a 1-min. count by entering 3000 in the Cycles box.

OH YE PITFALLS

Make backups of the working directories. Doing so will keep you from weeping if Visual C++ does something to your files. Be careful about erasing portions of an EZ-USB file you're modifying for your application. It's easy to erase too much code. It's much better to use forward slashes (//) to turn the line into a remark.

You also should be extremely careful when loading a program into the EEPROM. Because the USB software depends on the various identification bytes to choose the correct driver, changing the EEPROM could change the bytes. If you're lucky, the CD-ROM will contain another driver that will connect with your board when the first driver stops working. This happened to me. Now I have to load the monitor program manually. It used to load automatically at start-up.

When I attempted to compile the BulkXfer file, I received a message stating that the include file devioctl.h couldn't be opened. This is a Windows device I/O control header file that wasn't included in the EZ-USB software. You can find it in the Windows device developer's kit (WINDDK). The free DDK is included with the other files for this project.

APPLICABILITY

Setting up the USB port for A/D conversion enables a host of experiments. I use the system to acquaint my students with the basics of digital signal processing and digital filters. They can see the results of digitizing an analog signal as well as an immediate application in the FFT. Watching the frequencies change as they apply various inputs (signal generators, whistles, voices), students develop an intuitive understanding of the process. It's also a concrete example of how they can use Visual C++ for data manipulation and display. But, most of all, it shows them that it can be done.

Daniel Cross-Cole worked for the Department of Navy for 22.5 years as an electronics engineer. He retired in 2003 and is now a professor at DeVry University in Arlington, Virginia. Dan currently teaches courses about digital signal processing, telecommunications engineering, and computer peripheral devices. He enjoys ham radio and using Fast Fourier Transforms to characterize acoustic guitars. You may contact him at DCrossCole@dc.devry.edu.

Listing 2—*I used this routine to get data from the 8051's endpoint 8. Endpoint 8 data was transferred via pipe 3. The bulk transfer put the data into* `j_adcdata`. *I then transferred it to a buffer named "array."*

```
btc.pipeNum = 3;
// Read data in from EZUSB device
   bResult = DeviceIoControl (hDevice,
        IOCTL_EZUSB_BULK_READ,
        &btc,
        sizeof
(BULK_TRANSFER_CONTROL),
        j_adcdata,
        512,
        (unsigned long *)&nBytes,
        NULL);
   for (i = 1; i < 513; i++)
        Array[i]=j_adcdata[i];
```

PROJECT FILES

To download the code, go to ftp.circuitcellar.com/pub/Circuit_Cellar/2005/178.

RESOURCES

J. Axelson, *USB Complete*, Lakeview Research, Madison, WI, 2001.

H. Chamberlin's, *Musical Applications of Microprocessors*, Hayden Book Co., Rochelle Park, NJ, 1985.

J. Hyde, *USB Design By Example*, Intel Press, Santa Clara, CA, 2001.

SOURCES

EZ-USB Kit
Cypress Semiconductor Corp.
www.cypress.com

ADC0820 A/D converter chip
National Semiconductor
www.national.com

Build a Wi-Fi Web Server

Fred's hooked on the topic of wireless LANs. This month he shows you how to get a wireless web server up and running with an SRAM-equipped, ATmega128-based Wi-Fi module. Now you too can serve web pages from a wireless 802.11b station.

I'm in a hardware-building sort of mood this month. I've been reading nonstop since January. I figure it's now time to move from concept to design and realization. I thought I had it bad for Ethernet projects, but it seems that I'm eat up, to use a southern term, with interest for wireless LAN stuff as well.

If you're a veteran *Circuit Cellar* reader, you know that I started the 2005 season with a column about what I thought microcontroller-enabled 802.11b should look like. For the past few months, I've been concentrating on getting some additional scoop on 802.11b that I could turn into yet another working embedded wireless hardware device. Well, I'm now ready to share with you some of the wisdom I gained while studying in the faint light of the burning midnight oil.

This month I'll describe a finished piece of 802.11b hardware coupled with a working set of 802.11b firmware routines. As always, my goal for getting on a wireless LAN was to keep it all as simple as possible.

NEW WI-FI HARDWARE

As you can see in Photo 1, I stuck with the TRENDnet TEW-222CF CompactFlash 802.11b interface. I matched it up with an Atmel ATmega128L, which is the ATmega128's low-powered first cousin.

The ATmega128L has an extremely straightforward I/O interface. In this version of the hardware, I simply used whatever I/O pins I needed for interfacing to the CompactFlash NIC and left the rest for the outside world.

I also decided to add a fast 128K × 8 SRAM to the mix this time around. With the exception of the additional SRAM, there are no frills on the development board other than a standard RS-232 serial port and the Atmel ISP programming portal. I decided to include an AVR JTAG debugging/programming port to make it easier to debug and correct my coding mistakes. The JTAG interface also allows me to show you the details behind the code snippets in this column.

The Atmel AVR JTAG port is wired to support the JTAG ICE and the newer JTAGICE mkII. If you don't breath and sleep with Atmel microcontrollers, note that the JTAGICE mkII is identical in operation to the legacy JTAG ICE, but it also includes support for the new single-wire debugWIRE interface found on the smaller, low-pin-count AVR microcontrollers. The JTAGICE mkII also supports USB communication between it and the AVR Studio IDE, which makes the debugging cycles move much faster.

I took a lot of flack in the AVR forums for the way in which I implemented the 64K × 16 SRAM IC on the ATA hard drive controller. Instead of using the ATA hard drive controller ATmega128's built-in external memory management scheme, I tied the controller's big SRAM directly to the 16-bit bus on the IDE hard drive. The idea was to capture the outgoing IDE data in the fast SRAM without too much intervention by the ATA hard drive controller's ATmega128 microcontroller. The downside to doing things my way was that I destroyed the programmer's ability to use the AVR's external memory management system.

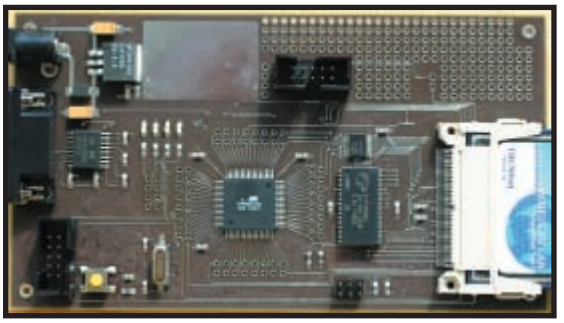


Photo 1—*I stretched the original universal Wi-Fi design to make room for the 128K × 8 SRAM and its companion address latch. I also added an official six-pin I²C portal. Cutting jumpers on the bottom side of the board can disconnect the four general-purpose LEDs from their respective I/O pins. After the LEDs are disconnected via the jumper removal, you can add some pins and jumper blocks to reconnect them to the ATmega128L's I/O pins.*

My proprietary SRAM connections didn't deter many of the hardcore AVR coders. They went around my one-of-a-kind SRAM riggings and turned their ATA hard drive controllers into a multitude of other SRAM-equipped devices. I'm pretty sure that the BASCOM-AVR guys had the ATA hard drive controller serving web pages and running an AVR version of Bill Gates's FAT16.

I didn't use the 128 × 8 SRAM device my way on the new AVR-based 802.11b wireless module. Instead, I wired the big SRAM as two 64-KB banks of external

SRAM. The SRAM's A16 address line, which is connected to I/O pin PD6 on the ATmega128L, controls SRAM bank selection. Because the ATmega128L contains an internal 4-KB block of SRAM, normal addressing methods will allow only 60 KB of the 64-KB bank of SRAM to be accessed.

If you take a close look at the ATmega128L datasheet, you'll see that there are masking methods that you can use to fake out the SRAM's address lines to get the full use of the 64-KB bank of SRAM if your project requires it. Using the A16 banking I/O line is easier and safer. Plus, using the banking I/O pin gives you 120 KB of usable external SRAM.

My first inclination was to use the leftover SRAM from the ATA hard drive controller project. However, the wireless module's electronics and the Wi-Fi card all run at 3.3 VDC. My roll of ATA SRAM contained 5-VDC parts, so I had to hit the datasheets and select a suitable SRAM IC and address latch. I finally settled on the Cypress CY7C1019CV33 SRAM IC, which is the slowest part of the bunch. But it's plenty fast, even with its 15-ns access time. I chose the 12-ns part, which has an access time that's well below my calculated AVR t_{RLDV} (read low to data valid) value of 76 ns. I came up with 76 ns using an algorithm that I gleaned from the ATmega128L datasheet:

$$t_{RLDV} = 1t_{CLCL} - 60 \text{ ns} = 76 \text{ ns}$$

where the Wi-Fi module AVR oscillator frequency equals 7.3728 MHz and

$$t_{CLCL} = \frac{1}{7.3728} \text{ MHz} = 136 \text{ ns}$$

With the CY7C1019CV33's 12-ns access time, it's obvious that wait states will be unnecessary when accessing the external SRAM.

The ATmega128L's datasheet also voices a concern about the address latch. The ATmega128L's external memory interface was designed for use with a 74AHC series latch. My latch decision was based on 3.3-VDC operation versus the address latch guidelines in the datasheet. I'm partial to 573 latch layouts, so I eliminated latches with a 373 designation. I ended up selecting the Texas Instruments 3.3-VDC SN74LVTH573PWR in the 20-TSSOP package. Compared to the address latch, the CY7C1019CV33 is a relatively large part; it's packaged as a 32-lead 400-mil molded SOJ. I would've preferred the SRAM in the smaller TSOP II form, but I couldn't obtain that package scheme from my SRAM distributor. Mick Jagger is right. You can't always get what you want.

The ATmega128L guarantees a minimum address hold time of 5 ns after the ALE signal falls low. The datasheet warns that you must choose the address latch accordingly. As it turns out, the SN74LVTH573PWR needs only a 1.5-ns window to capture the address data. You're covered.

Another critical factor concerning the selection of the address latch is the address-valid set-up time (t_{AVLLC}) before the ALE signal drops low. The minimum t_{AVLLC} value with a 7.3728-Hz clock and 3.3 DC operating voltage is:

$$t_{AVLLC} = 0.5t_{CLCL} - 10 = 58 \text{ ns}$$

You're covered here too because the SN74LVTH573PWR needs only 0.7 ns of set-up time. You also have the ALE pulse width. The minimum ALE pulse width of the ATmega128L configuration with a 7.3728-MHz clock is 121 ns. The SN74LVTH573PWR needs only 3 ns of the ALE pulse width.

The timings assume that the capacitance on the bus lines isn't excessive. So, I checked the capacitance loading figures as best I could. I couldn't get a handle on the Wi-Fi card because the CompactFlash card datasheet didn't have any in-depth electrical specifications. However, according to the SRAM datasheet, the CY7C1019CV33 comes in at only 8 pF. There really aren't enough devices on the address bus or data bus to throw the capacitance loading into high gear. So, I felt pretty safe with the designated address latch/SRAM combination.

The SRAM system design is flexible because you can choose to use the ATmega128L's external SRAM control scheme or turn off the AVR automatic SRAM control subsystem. You can also bit bang data into the SRAM by manually manipulating the SRAM control lines and the address/data bus. The CompactFlash card and the SRAM can't operate concurrently when the ATmega128L's external SRAM is under the AVR's spell. However, it's possible to talk to both the CompactFlash card and the SRAM when the ATmega128L's external SRAM control bits are disabled. As you can see in Figure 1, the rest of the hardware is pretty simple stuff (the way I like it). Now let's cover the firmware behind the hardware.

FIRMWARE

The same types of Ethernet packets that flow across wire are almost identical to the Ethernet packets that float in thin air. That relationship makes it pretty easy for me to reuse some time-tested code from my Easy Ethernet series of hard-wired Ethernet devices. The Easy Ethernet code is basically a minimal TCP/IP stack with just enough oomph to get a message from one point to another by way of standard Internet protocols.

Easy Ethernet devices come in two microcontroller flavors, PIC and AVR. I used an ATmega128L microcontroller for this project. The Wi-Fi base code was ported from the Easy Ethernet AVR device firmware. I wrote the original Easy Ethernet AVR code with ImageCraft's ICCAVR. To keep things simple, I stuck with the ImageCraft C compiler for this project as well.

The Easy Ethernet AVR firmware is also the basis for the AirDrop-A 802.11b firmware, which is currently in operation. I pulled heavily from the AirDrop-A firmware, which had to be modified to accommodate the additional SRAM circuitry.

I'm dying to use my new JTAGICE mkII. Let's get started.

ON AIR

The purpose of this project was to put a minimal web server on the air with my new SRAM-equipped, AVR-based Wi-Fi module. The first obstacle was rethinking and rewriting all of the basic Wi-Fi I/O routines to support the additional SRAM and address latch. I then reworked all of my I/O port definitions to match the new SRAM-equipped hardware. Honestly, that's all I figured I'd really have to do because I was using the working AirDrop-A firmware's port. However, after staring at a dead Wi-Fi web server module, I performed some quick single-step debugging and determined that the basic CompactFlash card reads and writes from the original 802.11b code didn't work.

Listing 1 is a "before" and "after" compilation of the code needed to read and write the CompactFlash card's attribute registers. The CompactFlash card I/O control lines don't have any rivals in the "before" snippet of code. Adding the SRAM duplicated some of the I/O signal names like OE and

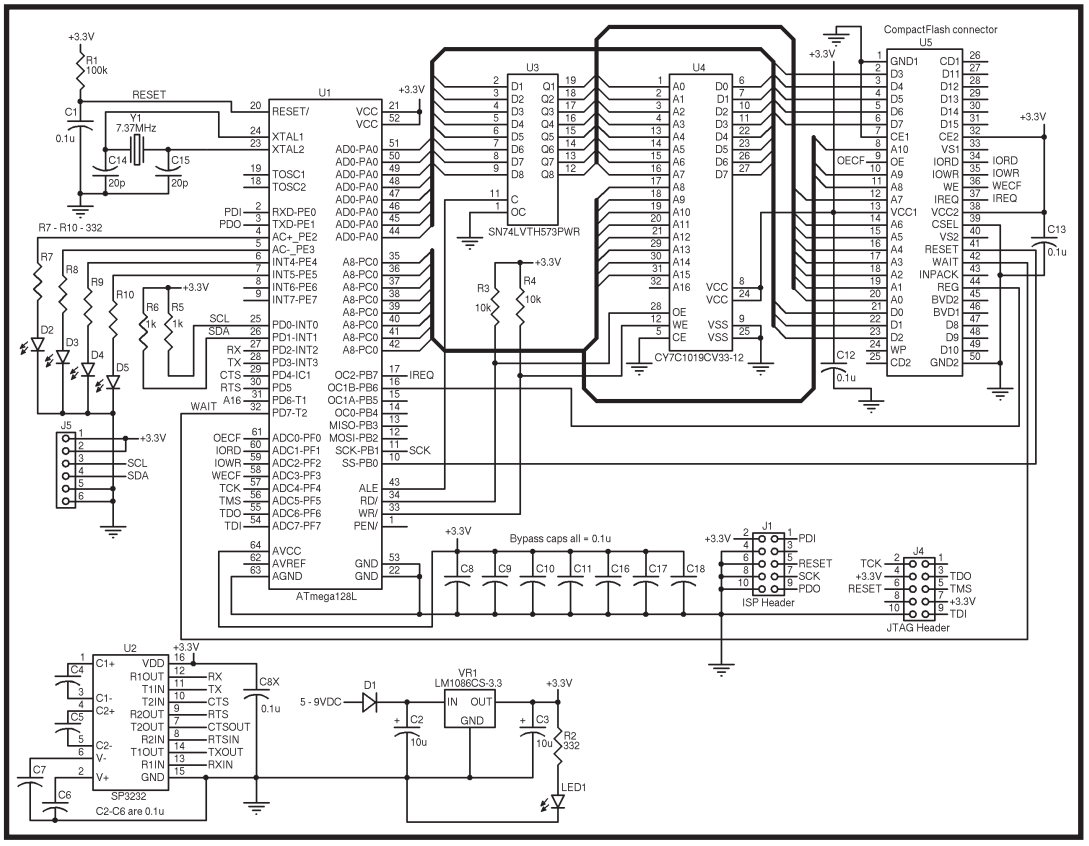


Figure 1—*The Wi-Fi web server hardware may be simple, but without the CompactFlash NIC and the 802.11b driver firmware, this would just be another standard ATmega128 configuration with external SRAM.*

WE. Thus, all of the I/O control labels that include the letters CF belong to the CompactFlash card. In addition, the "before" code always left a function with the data bus, which was separate from the address bus in the AirDrop-A scheme, in Input mode.

The old data bus state logic no longer applied after I added the 573 latch and combined what had been the data bus into a combination address/data bus. After additional debugging runs, I determined that the combined address/data bus I/O pins at the inputs of the address latch were actually in Input mode when the ATmega128L was attempting to write and latch the lower 8 bits of the target address.

Listing 1—*Adding the SRAM and address latch forced me to rethink my address and data bus logic.*

```
************************************************************************
Before
************************************************************************
void wr_cf_addr(unsigned int addr)
{
        addr_hi = (make8(addr,1)) & 0x07;
        addr_lo = addr & 0x00FF;
}
void wr_cf_reg(unsigned int reg_data,unsigned int reg_addr)
{
        TO_NIC;
        wr_cf_addr(reg_addr);
        data_out = reg_data;
        clr_WE;
        delay_ms(2);
        set_WE;
        FROM_NIC;
}
char rd_cf_reg(unsigned int reg_addr)
{
        char data,i;
        wr_cf_addr(reg_addr);
        clr_OE;
        i=1;
        while(—i);
        data = data_in;
        set_OE;
        NOP();
        return(data);
}
************************************************************************
After
************************************************************************
void wr_cf_addr(unsigned int addr)
{
    TO_NIC;
    set_ALE;
    addr_hi = make8(addr,1);
    addr_lo = make8(addr,0);
    clr_ALE;
}
void wr_cf_reg(unsigned int reg_data,unsigned int reg_addr)
{
        wr_cf_addr(reg_addr);
        data_out = reg_data;
        clr_WECF;
        delay_ms(2);
        set_WECF;
        FROM_NIC;
}
char rd_cf_reg(unsigned int reg_addr)
{
        char data,i;
        wr_cf_addr(reg_addr);
        FROM_NIC;
        clr_OECF;
        i=1;
        while(—i);
        data = data_in;
        set_OECF;
        NOP();
        return(data);
}
```

Now let's take a look at the "after" section of code. I still have to leave the combined address/data bus in Input mode when I leave a function. But the `TO_NIC` macro, which puts the combined address/data bus in Output mode, is only issued in the write address function (`wr_cf_addr`).

After squashing a few more minor bugs I found in my port I/O pin direction assignments, which are opposite the PIC I/O pin direction bits, I finally got the tuple hashing routines running successfully. Although the tuples can tell you important things about the CompactFlash card, I was only interested in making sure I got the correct I/O portal address so I could issue commands to the Wi-Fi card. The `cor_addr` variable holds the valid CompactFlash card's I/O portal address, which is 0x03E0 in this case. The 0x03E0 address is supposedly standard, but I still would rather determine the correct address on my own.

My troubles weren't over at that point. The Wi-Fi card enabled just fine, but that's where it ended. Using my wireless sniffer, I captured what I thought would be the Wi-Fi card's probe session with the access point (AP). However, I didn't see any radio activity from the TEW-222CF, which could have only meant that my CompactFlash card Wi-Fi register read and write routines were still in the ditch. After some additional debugging, I found that my Wi-Fi card wasn't receiving an initialization command. The `rd_cf_io16` and `wr_cf_io16` functions in Listing 2 (p. 38) weren't working as designed.

Before reading on, please download a copy of the 16-bit Wi-Fi card register read and write routines from the *Circuit Cellar* ftp site so you can follow along. I was only issuing the `TO_NIC` macro in the `wr_cf_addr` function, so the "before" `rd_cf_io16` function wouldn't work. The combination address/data bus was always in Output mode following the address write and latch operation. The cure here was the same medicine that helped me through the tuple hash. I issued only the `TO_NIC` macro in the address write/latch function and left the 16-bit read routines to call out the `FROM_NIC` input macro as they needed it.

After modifying the "after" code in the listing I asked you to download, the newly ported Wi-Fi code fired up and successfully joined the Florida room's LAN. At that point I was really anxious to play with the SRAM, but there was still the matter of running the simple web server.

CODING THE SERVER

The first step in getting the Wi-Fi web server off the ground is to make sure it's receiving the HTTP request correctly from the client. My Easy Ethernet/AirDrop-A TCP routines are good enough for simple Telnet transactions, but they weren't quite right for processing HTTP requests. I don't have enough space here to show you the entire TCP/IP function transformation, but you can download the Wi-Fi web server source code posted on the *Circuit Cellar* ftp site.

Basically, all I had to do was receive the HTTP request, parse the incoming HTTP request data, send an HTML page, and disconnect the TCP/IP session. I stopped the code in Photo 2a just before testing for the `GET /` string in the HTTP request I initiated from a PC. I pulled the first 32 bytes from the incoming HTTP request looking for only the first five characters. I sized the `http_temp` array to allow for parsing things following the initial `GET /` string.

For instance, following the `GET /` string, you could specify a file name to retrieve or include some codes to control or monitor I/O pins on the server. All I really wanted to do was set up a simple HTTP server framework and send some HTML to the client browser. After that was accomplished, I could add the code to serve a real web page and control or monitor stuff using the Wi-Fi web server's ATmega128L I/O.

Let's start at the beginning and fast forward to the breakpoint in Photo 2b. The Wi-Fi web server module initialized and joined the basic service set (BSS) controlled by the EDTP AP with a 192.168.0.151 IP address. The simple HTTP request (http://192.168.0.151) issued from a hard-wired PC wasn't equipped with Wi-Fi capability. A TCP/IP session was established between the PC client and the Wi-Fi web server via the EDTP AP. After the TCP/IP session was established, the client could exchange data with the Wi-Fi web server. The client PC then issued the HTTP request via its web browser application. The Wi-Fi web server received the HTTP request data, determined that it was addressed to 192.168.0.151 port 80 (the well-known HTTP port), and parsed the incoming HTTP data looking for the string.

Normally, if all goes as planned, the Wi-Fi web server will find the `GET /` string. It will then push some HTML into its transmit buffer, tack on a FIN in the TCP header, and transmit the HTTP reply back to the PC client.

Photo 2b shows the HTTP response loaded in the Wi-Fi web server's transmit buffer. Where did that come from? Take a look at Listing 3. That's my web page complete with the HTTP high sign. This entire thing started and ended with the client web browser shown in Photo 3. When you look over and follow the web server thread in the Wi-Fi web server source code, you'll see that it really didn't take much coding effort to put that one-liner in the client's web browser window.

Listing 2—*It didn't matter if the write functions worked well because the read functions were being sabotaged by the addition of the SRAM and address latch.*

```
char  send_command(unsigned int cmd, unsigned int parm0)
{
       char cmd_code,rc;
       unsigned int cmd_status,evstat_data;
       do{
           cmd_data = rd_cf_io16(Command_Register);
         }while(cmd_data &  CmdBusy_Bit_Mask);
       wr_cf_io16(parm0, Param0_Register);
       wr_cf_io16(cmd, Command_Register);
       do{
          cmd_data = rd_cf_io16(Command_Register);
         }while(cmd_data &  CmdBusy_Bit_Mask);
       do{
           evstat_data = rd_cf_io16(EvStat_Register);
          }while(!(evstat_data & EvStat_Cmd_Bit_Mask));
       cmd_status = rd_cf_io16( Status_Register);
       cmd_code = cmd_status &  Status_CmdCode_Mask;
       rc = cmd_status &  Status_Result_Mask;
       wr_cf_io16( EvStat_Cmd_Bit_Mask, EvAck_Register);
       switch(rc)
       {
              case 0x00:
                     rc = 0;
                     break;
              case 0x01:
                     rc = 1;
                     break;
              case 0x05:
                     rc = 2;
                     break;
              case 0x7F:
                     rc = 3;
                     break;
              default:
//Return_code = result_code;
                     break;
}
return(rc);
```

Listing 3—*Everything in this super-simple HTML line between the <html> and </html> tags can be found in the browser window shown in Photo 3.*

```
const char http_ok[]="HTTP/1.1 200 OK \r\nContent-Type: text/html
\r\n\r\n<html><head><title> EDTP Wi-Fi Server
PAGE</title></head><body><b><u>WI-FI WEB
SERVER CODE DOESN'T HAVE TO BE COMPLICATED</b></u></body></html>";
```

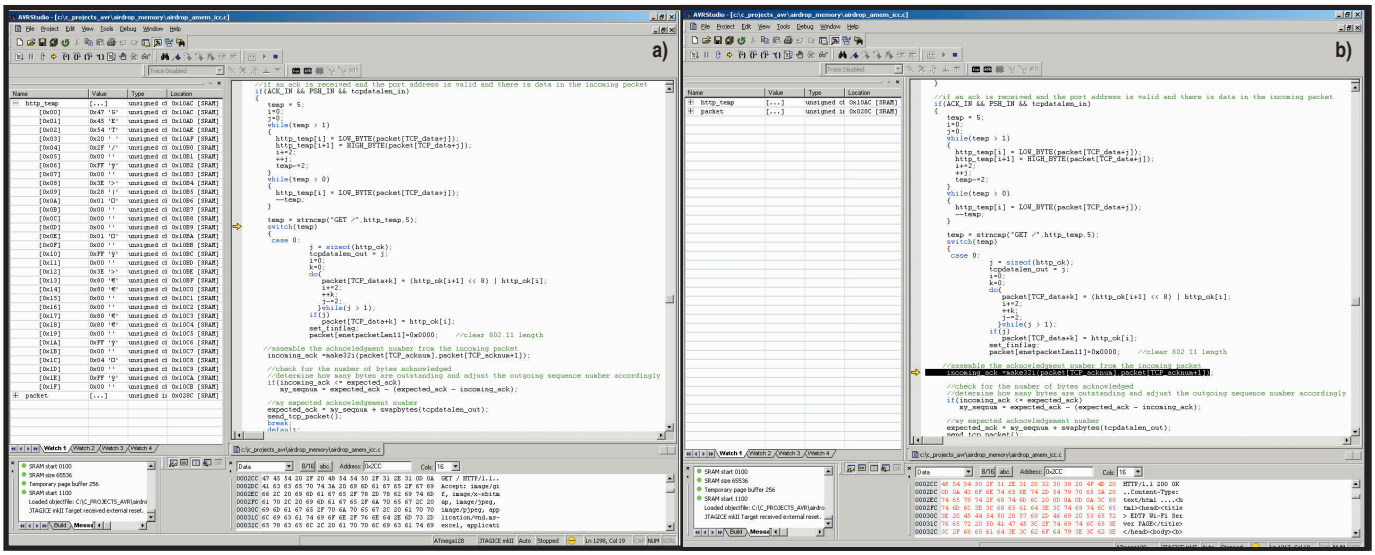


Photo 2a—*The GET / string is definitely in the Wi-Fi module's receive buffer. If all goes as planned, the next screen shot should show the HTML that's to be sent to the client PC.* **b—***The correct response is shown in the hex dump of the transmit buffer contents. The web page here is simply a line of HTML that's stored in the ATmega128L's program flash memory.*

128K × 8 SRAM

Now that I've shown you how easy it is to create a simple wireless web server, you can add your personality to the code and generate pages yourself. You'll want to store stuff along the way, but if you look closely at the data memory layout in the Wi-Fi source code, you'll see that the Wi-Fi web server's little TCP/IP stack uses most of the ATmega128L's on-chip SRAM. I included the 128K × 8 SRAM on the Wi-Fi web server board so you don't have to limit your web server application because of a lack of available data memory.

The way you access the external SRAM depends on how your assembler or compiler handles the extra memory area. You may download a screen shot of a JTAGICE mkII debug window that shows what's behind the ImageCraft external SRAM handler mechanism. The ImageCraft scheme allows you to allocate chunks of external SRAM space.

I carved out a 256-byte block of external SRAM (xmem) beginning at data memory address 0x1100, which is the beginning of the external SRAM memory area and immediately follows the ATmega128L's 4 KB of on-chip SRAM. The real work is done with the in-line assembler routine, which I generated with the ICCAVR Application Builder. All I had to do was designate the external SRAM address, name the memory block, and enter the number of bytes to reserve. My hand-written code simply cleared the SRAM block to zero and then wrote the block with ascending ASCII characters beginning at ASCII 0x00. Nothing to it.

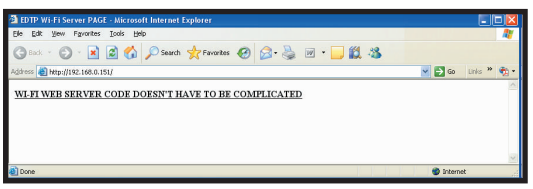


Photo 3—*In my opinion, building good-looking embedded web pages is more difficult than writing Wi-Fi web server code.*

ACK FIN

Although there's plenty of SRAM to go around, you'll probably want to store your large web pages in program flash memory. I noticed that when I told the ImageCraft C compiler that I had 64 KB of external SRAM, it took it upon itself to start storing variables there. There's a warning that comes up when you tell the compiler you have the extra data memory. It basically says that if you want to control the external SRAM's content, don't tell the compiler about it.

As the title of this section implies, I've reached the end of my paper ration. The basics of serving web pages from a wireless 802.11b station are in place. You now have the tools necessary to build your own Wi-Fi web server application. So, remember to keep it simple. It doesn't have to be complicated to be embedded.

Fred Eady has more than 20 years of experience as a systems engineer. He has worked with computers and communication systems large and small, simple and complex. His forte is embedded-systems design and communications. Fred may be reached at fred@edtp.com.

PROJECT FILES

To download the code, go to ftp.circuitcellar.com/pub/Circuit_Cellar/2005/178.

RESOURCE

Atmel Corp., "8-bit Microcontroller with 128K Bytes In-System Programmable Flash: ATmega128/L," rev. 2467MA-AVR, 2004.

SOURCES

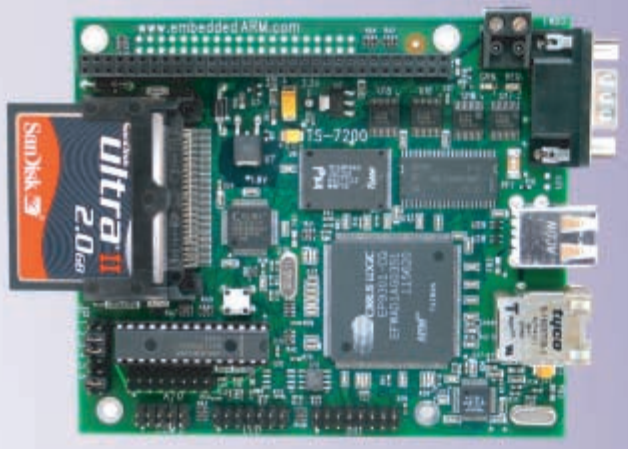
JTAGICE mkII and ATmega128L MCU
Atmel Corp.
www.atmel.com

TRENDnet TEW-222CF
TRENDware
www.trendnet.com

CY7C1019CV33 SRAM
Cypress Semiconductor
www.cypress.com

ICCAVR C Compiler
ImageCraft
www.imagecraft.com

200 MHz ARM - PC/104



Shown with optional A/D converter, Compact Flash and RS-485

TS-7200 $149 $119 qty 100

- 32 MB SDRAM (64 MB optional)
- 8 MB Flash (16 MB optional)
- Compact Flash socket
- 10/100 Ethernet
- 2 USB ports
- 20 Digital I/O, A/D options
- 2 Serial Ports
- Optional USB WiFi adapter
- Support for Windows® CE

- EP9302 ARM9 processor with MMU
- Boots TS-Linux from on-board flash in 5 seconds
- TS-Linux for ARM is open source GNU/Linux
- Low power +5v @ 400mA ▪ 3.8" x 4.5"

Linux 2.4 NetBSD

Windows is a registered trademark of Microsoft Corporation

200 MHz ARM - PC/104

Shown with optional A/D converter, USB Flash drive and RS-485

TS-7250 $149 $119 qty 100

- 32 MB SDRAM (64, 128 MB optional)
- 32 MB Flash (128 MB optional)
- 10/100 Ethernet
- 2 USB ports
- 20 Digital I/O
- 2 Serial Ports
- 5 ch. 12 bit A/D (optional 13 channels)
- Optional USB WiFi adapter

- EP9302 ARM9 processor with MMU
- Boots TS-Linux from on-board flash
- TS-Linux for ARM is open source GNU/Linux
- Low power +5v @ 400mA ▪ 3.8" x 4.5"

The 256 MB USB drive shown holds our Debian 3.0 self-hosting development environment for ARM. Booting to the development file system allows code development to take place directly on the target hardware eliminating any issues with transfering newly compiled files from the desktop to the target device.

Cross-compiler tools are also available for Windows and Linux.

www.embeddedx86.com

(480)-837-5200
16610 E. Laser Drive #10
Fountain Hills, AZ, 85268 USA

Technologic SYSTEMS

133 MHz 5x86 - PC/104

TS-5600
$279

- PC/104
- PCMCIA slot
- 10/100 Ethernet
- SPI bus
- 2 Serial Ports
- Compact Flash
- 24 Digital I/O
- 12-bit 8 chan. A/D option

- AMD Elan 133 MHz 586 with 32 MB RAM (64 MB optional)
- PC compatible: BIOS, DOS, RTC, I/O & memory map
- TS-Linux 3.0 is open source GNU/Linux - Glibc 2.2.5
- Power +5v @ 830mA
- 4.3" x 5.6"

Shown with optional flash modules A/D, 3S-485 and Merlin cellular modem

33 MHz 386 - PC/104

TS-3400
$159

- PC/104
- Low EMI
- DiskOnChip
- DiskOnChip
- 37 Digital I/O
- 2 Serial Ports

- 33 MHz Intel 386EX with 8 MB RAM (16 MB optional)
- DOS-ROM or TS-Linux pre-installed
- PC compatible: BIOS, DOS, RTC, I/O & memory map
- 1 MB flash drive
- Power: +5v @ 500 mA

Shown with optional flash module and RS-485

PC/104 Ethernet Adapter

TS-ETH10 **$49**

- PC/104 daughter board
- 10baseT Ethernet
- DOS, Linux and NetBSD drivers
- Link and activity LED
- Header for external LED hookup
- 3.5" X 1.6"
- Jumper selection allows up to 4 boards per system

Shown with optional 16 bit connector

Technologic SYSTEMS

- PC/104 Computer Boards
- Enclosures
- PC/104 Peripherals
- Custom Designs
- Exceptional Support

PC/104 Peripherals

Enclosures $85
Metal enclosure for Single Board Computers provides regulated +5 VDC from 12-38 VDC input power. Connections available at back panel. Suitable for office or factory.

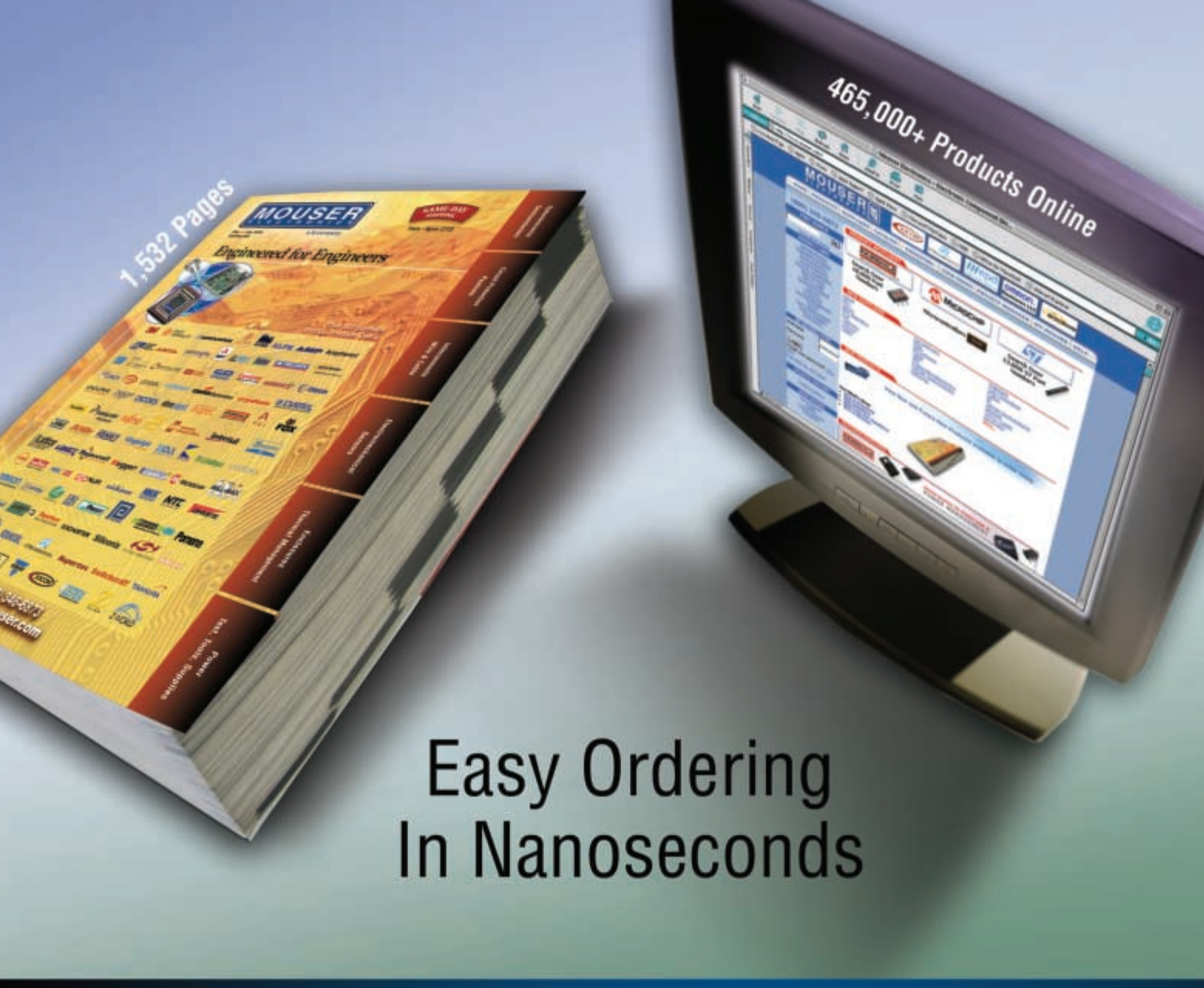
TS-SER4 $89
Four PC compatible COM ports using the 16550 UART with 16-byte FIFOs, full handshaking on two ports, RS-485 option on two ports, jumper configurable.

TS-DIO24 $39
24 0-5V DIO lines, opto-22 compatible 50 pin connector, 4 outputs can drive up to 30V @1 Amp, 4 inputs are IRQ capable, I/O address jumper selectable.

TS-Modem $129
33.6K baud, jumper configurable as COM1-6, supports IRQs 3-15, V.32 V.42 compliant, standard AT commands, caller ID, 56K baud optional.

TS-Linux v3.0

- Introducing TS-Linux version 3.0 Our latest distribution is built from open-source featuring a 2.4.18 kernel, Apache web server, FTP client/server, bash, etc. With a footprint of 15 MB or less this is a great place to start your application.
- The TS development environment compiles application code using the libraries and headers installed in TS-Linux. Applications developed on your desktop computer copy directly.
- Our User Mode Linux patched kernel runs TS-Linux 3.0 kernel and file system as an application on your desktop computer during development.



Easy Ordering In Nanoseconds

Delivering the latest with the greatest.

It's easy to order from our catalog via phone or online — in nanoseconds. Mouser is the catalog distributor focused on the needs of engineers. We deliver the latest technologies with the great service and support you need.

SEMICONDUCTORS | PASSIVES | INTERCONNECTS | POWER | ELECTROMECHANICAL | TEST, TOOLS & SUPPLIES

New Products
New Suppliers
New Technologies
New Catalog Every 90 Days!

MOUSER ELECTRONICS
a tti company

(800) 346-6873
www.mouser.com

Mouser® and Mouser Electronics® are trademarks of Mouser Electronics, Inc.

Digital RC Servo Controller (Part 2)

Circuitry Details

In the first part of this series, Eric introduced you to the theory behind RC serial servo control. This month he describes the design of a high-resolution, 32-channel RC servo controller with a serial interface.

Last month I explored the principles behind RC serial servo controllers. I also described a true hardware-based, 32-channel digital serial servo controller architecture that's well suited for demanding multichannel RC servo control applications. Unlike all of the other commercial offerings, the design uses dedicated parallel hardware resources for PWM pulse train generation at 16-bit accuracy and 12-bit resolution with all 32 channels fully synchronized! This month I'll describe the circuitry.

CIRCUITRY UP CLOSE

Figures 1 and 2 (p. 44) show the digital servo controller circuitry. Note that the 32 channels were partitioned into four banks of eight. Because of the potentially large currents flowing through the numerous servos, it's important to carefully plan the servo power and ground return paths. In fact, large currents flowing through the PCB can result in ground bounce and servo chatter. Each isolated bank of eight channels should have solid power and ground copper planes as well as a power connector mounted on the side. This configuration will enable you to connect heavy gauge wiring in a star pattern from each bank to the power supply, which of course must be able to handle the system's peak current demands.

The power connector to the MCU and electronics is set aside, so you power it from a clean power source separate from the servo power. If you don't have a separate supply for the electronics, you should add an appropriately sized electrolytic filter capaci-

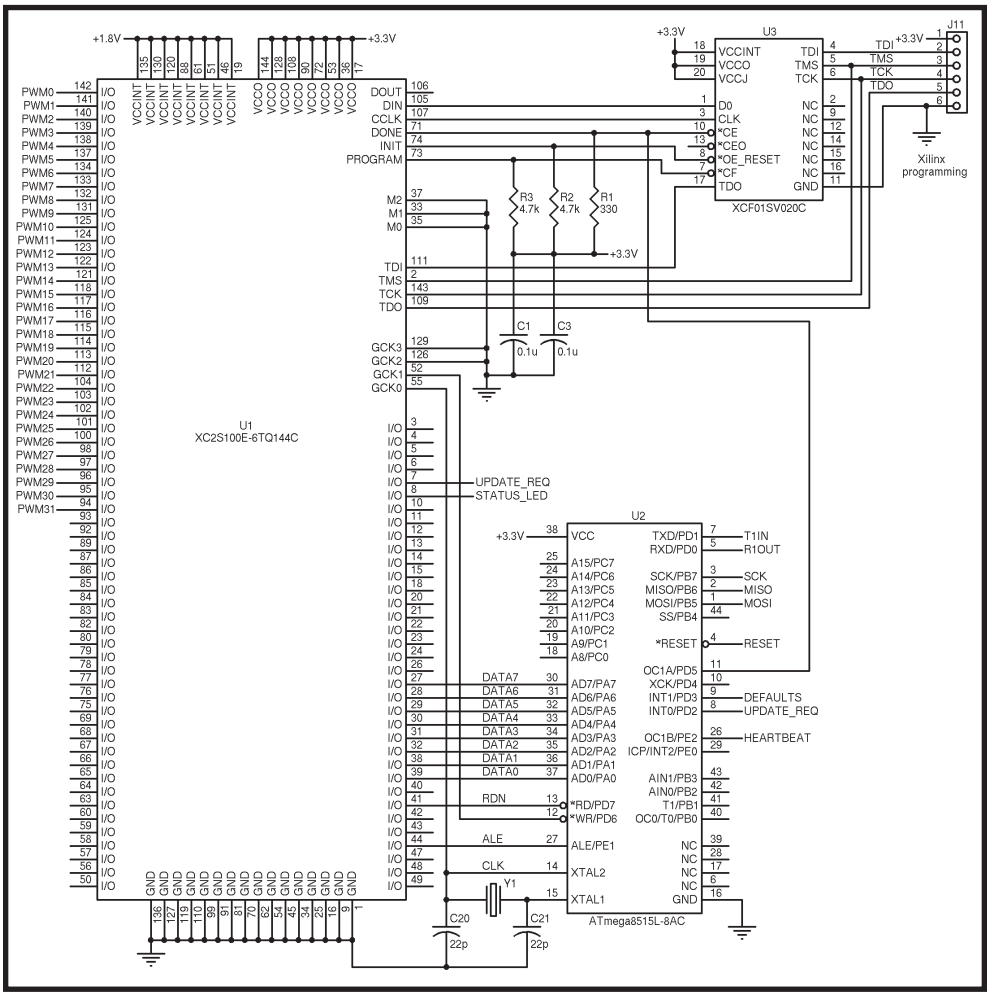


Figure 1—*The heart of the system includes an Atmel microcontroller with an external memory bus interfaced to a Xilinx FPGA. The FPGA's 32 PWM generation units have control registers that are memory-mapped in the MCU's external address space. A flash memory-based program memory automatically loads the FPGA code at power-up.*

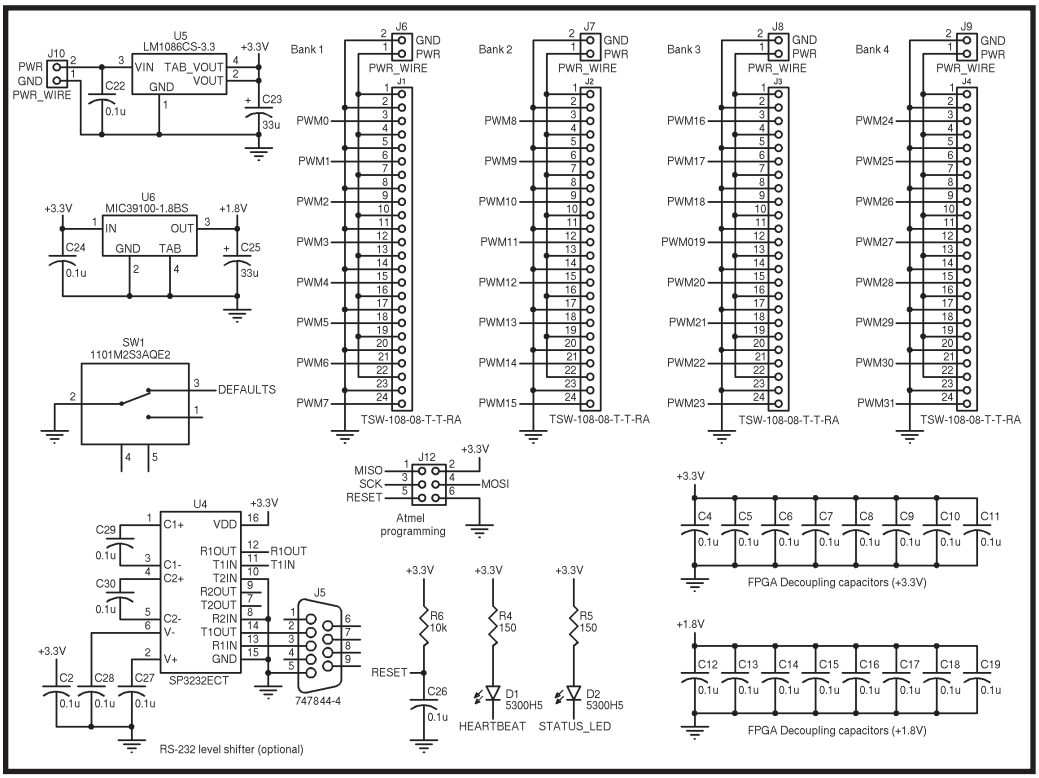


Figure 2—*The 32 servo channels were organized into four banks of eight because of the potentially large currents flowing through the servos. Each bank has a local power supply connector that enables external star ground and power distribution. The power connection to the electronics also has been separated out.*

tor across the MCU's power input connector.

TRUE CONCURRENCY

A 100,000-gate Xilinx Spartan IIE FPGA houses the 32-parallel PWM generation modules. Each module has a 16-bit register pair (MSB/LSB) that specifies the respective pulse widths (12 bits are used). The resulting group of 64-byte registers is memory mapped to an external memory bus. Figure 3 is a block diagram of the FPGA's internal architecture.

I created the FPGA design in VHDL with Xilinx ISE tools. The master 16-bit counter on top is driven from the 4-MHz system clock, which is conveniently generated externally by the ATmega8515L microcontroller. A bank of 32 independent PWM generation modules is synchronized to the master counter.

The UPDATE_REQ signal is triggered at the master counter's quarter count mark. It's used to generate an interrupt to the MCU to indicate that it can safely reload the FPGA's 64 memory-mapped control registers. This way you don't have to update the registers during the pulse generation phase, which can potentially generate runt pulses.

An 8-bit system configuration register, ctrl_reg, allows you to select certain system options. First, the 32 PWM channels are organized into four logical banks that can be individually disabled through this register. In addition, a global option allows you to turn off each PWM signal completely when you send a 0 command to the channel. This makes the servos go limp so you can move them manually. When this option is disabled, a 0 command value will cause the servo to hold at position 0. An LED is also tied to one of the ctrl_reg bits. It simply blinks when a valid serial command packet has been received, which is useful for troubleshooting.

The individual PWM modules enable the generation of a low-jitter, high-accuracy PWM waveform. A VHDL code snippet in Listing 1 describes the operation of one PWM channel. The same module is simply replicated 32 times inside the FPGA.

The PWM module's operation is straightforward. It simply compares its target pulse-width register value against the master counter value and sets the PWM output to zero when the master counter exceeds this value.

One thing to note is that even if the commanded pulse-width register pwm_val is logically defined as a 2-byte register (MSB/LSB), only the top 12 bits are used. This greatly

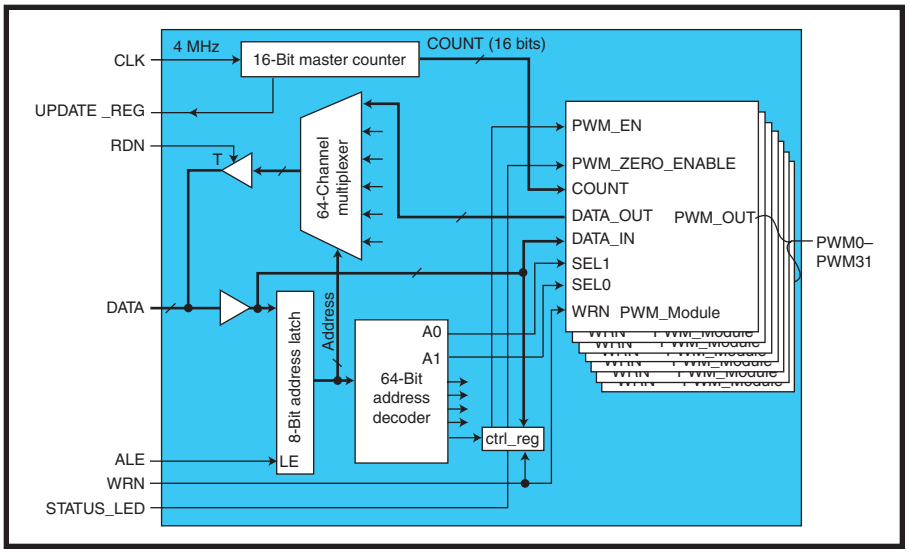


Figure 3—*Each of the 32 PWM generation modules in the FPGA has a memory-mapped 16-bit register (2 bytes) used to set the pulse width. General configuration register ctrl_reg is used to set system options such as bank enable and zero value turn-off. At the top is the 16-bit master counter to which all the modules are synchronized.*

Listing 1—*In this VHDL code snippet of a single channel PWM generation module,* `count` *is an external 16-bit master counter operating at 4 MHz. Each module has a 2-byte* `pwm_val` *register for storing the pulse-width count. To simplify the logic and reduce the FPGA's size, the lower 4 bits of the LSB register are ignored to give an overall 12-bit effective resolution.*

```
//12-bit register logic
process(wrn, sel0, sel1, data_in)
begin
  if (wrn'event and wrn='1') then  //On rising edge of wrn,
                                   //then latch data
    if (sel1 = '1') then     //If sel1 high, then MSB (has priority)
       pwm_val(11 downto 4) <= data_in;
    elsif (sel0 = '1') then  //If sel0 high, LSB (ignore low nibble)
       pwm_val( 3 downto 0) <= data_in(7 downto 4);
    end if;
  end if;
end process;
//Data out values
data_out <= pwm_val(11 downto 4) when (sel1='1') else //Highest priority
                                                      //sel1 (MSB)
  (pwm_val( 3 downto 0) & "0000");    //in case both high or sel0 high
//Output PWM logic
pwm_out  <=  '0' when (pwm_en = '0') else     //If "0=off" mode enabled
                                              //and value = 0
       '0' when (pwm_val = "000000000000000" and pwm_zero_enable = '1') else
       '1' when (count > "1111100000110000") else     //Count of 63536
       '0' when (count > ("000" & pwm_val(11 downto 0) & "0")) else
//Multiply value by 2
       '1';
//Shift left by one for 180° servo control mode (deltaT = 2 ms)
```

reduces the FPGA's design size. It's still higher resolution than the 10-bit digital servos can resolve. The reason for ignoring the bottom 4 bits of the LSB instead of the top 4 bits of the MSB is that it's still possible to command the servo with only the 8-bit MSB for backwards compatibility with the 8-bit Mini SSC-II protocol. The end result is a programmable pulse that varies between 0.5 and 2.5 ms at 12-bit resolution and 16-bit accuracy. Note that the initial 0.5 ms common to each PWM cycle is generated at the end of the period for simplicity instead of at the beginning. But this produces the same result because the signal is periodic.

ATmega8515L CONTROL

The ATmega8515L MCU runs the system. I developed all the firmware in C language using an ImageCraft ICCAVR compiler.

The low-cost ATmega8515L has many features that make it well suited for this application. It has 8-KB flash memory, 512 bytes of SRAM, and a convenient 512 bytes of onboard EEPROM. It also has an external memory bus that's useful for interfacing with the FPGA. In addition, the ATmega8515L, which runs on 3.3 V, has external interrupts and a UART. During development, the firmware was downloaded to the ATmega8515L through the on-chip-debug interface.

Figure 4 is the ATmega8515L's firmware flowchart. At start-up, the ATmega8515L's internal registers and watchdog are initialized. The ATmega8515L then waits for the FPGA to finish programming from its external boot memory. Following this, the ATmega8515L reads and sets all the user-configurable settings from its internal EEPROM (e.g., speed settings, servo limits, and the default start-up position of all servos).

The firmware then proceeds with the main thread of execution. The first step consists of checking if any incoming serial characters are waiting in the serial buffer, which is filled asynchronously by the UART's receive interrupt service routine (ISR). If characters are waiting, then they're read out one at a time and parsed with the serial protocol state machine. If an entire packet is received with a valid checksum, the command is executed.

The next step involves checking to see if the refresh_fpga flag is set. Only the INT0 ISR can set this flag. The FPGA generates the signal externally. After the flag is set, it's time to compute the next positions for all 32 channels. When velocity loops are enabled, the intermediate interpolated positions are computed until the target position is reached. This generates smooth interpolated motion and reduces servo twitching.

Some positions may have been changed by external requests received through the serial port. Servo bounds checking is performed on each channel, and finally the entire array of current servo position values is transferred to the memory-mapped FPGA registers as a simple array-to-array memory transfer.

The final step involves housekeeping tasks such as kicking the watchdog. Also, as a special touch, instead of the traditional blinking heartbeat LED, I decided to control this LED through one of the MCU's internal PWM pins (OC1B). Changing the PWM value can modulate the LED's

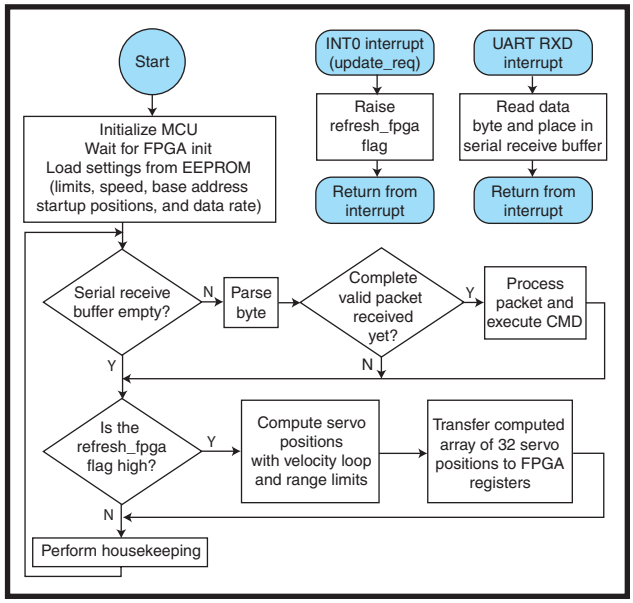


Figure 4—*One main thread of execution is responsible for processing and executing incoming serial commands as well as updating the FPGA registers with fresh values computed by the velocity loop (with bounds checking). The UART's interrupt service routine simply buffers any incoming characters. The external interrupt service routine raises the refresh_fpga flag.*

brightness. By implementing two consecutive intensity ramps with different time constants, I was able to generate a really cool heartbeat effect. When powered, the board truly appears alive!

FOLLOWING PROTOCOL

A serial communications protocol was developed to be compatible with the Mini SSC-II so that existing commercial software could be used. A number of modifications were performed to the original protocol to enable bidirectional communications to and from the board.

The existing protocol allows the use of 0xFF only as the synchronization character. This limitation had to be removed to allow the transfer of arbitrary data values. The restriction is preserved on the address range (0x00-0xEF). Because of this compromise, I designed the serial parser state machine with the ability to resynchronize itself automatically if it receives a 0xFF during the address slot, should the state machine get out of sync.

Let's take a look at an explicit example. The serial protocol is the following: [sync 0xFF][servo address 0x00-0xFE][servo position 0x00-0xFF]. First, the serial parser state machine must look for the sync token 0xFF. Then, it must wait for the servo address and servo position.

Now let's assume the worst-case scenario in which the commanded servo position is actually equal to the sync value 0xFF. Let's also assume that the parser state machine has gone out of sync and has taken the 0xFF servo position value to be the sync byte of a new message. In this sort of scenario, the next logical character to be read will be 0xFF, which is the legitimate header of the next message. At that point, the parser state machine would expect a servo address and would read 0xFF, which is defined as an invalid address. It would simply ignore it and wait for the next byte to be the address, thereby resynchronizing with the datastream.

Another change to the protocol involved devising a backwards-compatible technique to transfer board configuration data as well as commands to and from the target board. A simple solution involved reserving the absolute servo address range 0xF0 to 0xFE as representing the 15 bytes of a broadcast command packet. A full command set was developed as shown in Table 1.

The commands can take as few as 2 bytes and as many as 12 bytes of the 15 reserved bytes. A checksum character is always required at address 0xFE (all unused addresses must be written with 0x00). To send a command, simply fill in all the required command data values by sending a sequence of servo position commands with servo addresses ranging from 0xF0 to 0xFD. As the last step, send the checksum character at address 0xFE. This will trigger the execution of the command. Because all the boards in the optional

Servo channel address		**0xF0**	**0xF1**	**0xF2**	**0xF3**	**0xF4**	**0xF5**	**0xF6**	**0xF7**
		0	1	2	3	4	5	6	7
CMD_FORCE_REBOOT	Command	dest addr	0x10	0x00	0x11	0x22	0x33	0x44	0x55
	Reply	(none)	(none)	(none)	(none)	(none)	(none)	(none)	(none)
CMD_PING	Command	dest addr	0x11	0x00	0x00	0x00	0x00	0x00	0x00
	Reply	board addr	0x11	0x01	0x00	0x00	0x00	0x00	0x00
CMD_GET_INFO	Command	dest addr	0x12	0x00	0x00	0x00	0x00	0x00	0x00
	Reply	board addr	0x12	BOARD TYPE	FIRMWARE REV	FPGA REV	CONFIG_BYTE	0x00	0x00
CMD_READ_MEM	Command	dest addr	0x13	ADDR0	ADDR1	0x00	0x00	0x00	0x00
	Reply	board addr	0x13	ADDR0	ADDR1	DATA0	DATA1	DATA2	DATA3
CMD_WRITE_MEM	Command	dest addr	0x14	ADDR0	ADDR1	DATA0	DATA1	DATA2	DATA3
	Reply	board addr	0x14	ADDR0	ADDR1	0x00	0x00	0x00	0x00
CMD_GET_POS	Command	dest addr	0x15	BANK (0–7)	0x00	0x00	0x00	0x00	0x00
	Reply	board addr	0x15	BANK (0–7)	POS0	POS1	POS2	POS3	POS4
			0xF8	**0xF9**	**0xFA**	**0xFB**	**0xFC**	**0xFD**	**0xFE**
			8	9	10	11	12	13	14
CMD_FORCE_REBOOT	Command		0x66	0x77	0x88	0x99	0xAA	0xBB	checksum
	Reply		(none)	(none)	(none)	(none)	(none)	(none)	(none)
CMD_PING	Command		0x00	0x00	0x00	0x00	0x00	0x00	checksum
	Reply		0x00	0x00	0x00	0x00	0x00	0x00	
CMD_GET_INFO	Command		0x00	0x00	0x00	0x00	0x00	0x00	checksum
	Reply		0x00	0x00	0x00	0x00	0x00	0x00	checksum
CMD_READ_MEM	Command		0x00	0x00	0x00	0x00	0x00	0x00	checksum
	Reply		DATA4	DATA5	DATA6	DATA7	0x00	0x00	checksum
CMD_WRITE_MEM	Command		DATA4	DATA5	DATA6	DATA7	0x00	0x00	checksum
	Reply		0x00	0x00	0x00	0x00	0x00	0x00	checksum
CMD_GET_POS	Command		0x00	0x00	0x00	0x00	0x00	0x00	checksum
	Reply		POS5	POS6	POS7	0x00	0x00	0x00	checksum

Table 1—*The command set added to the Mini SSC-II protocol allows you to send commands and read back configuration data from the target board. The process simply involves sending a sequence of conventional position commands at the specific servo addresses shown with the data values indicated. Sending the last checksum character triggers the command execution and the reply.*

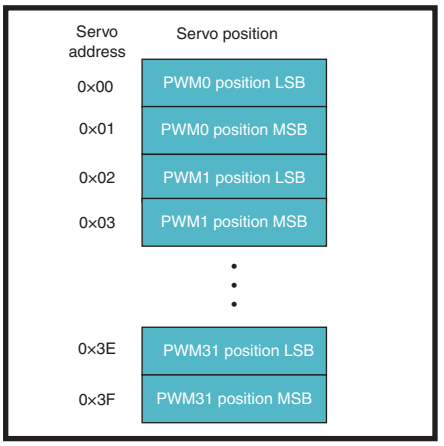


Figure 5—*The PWM register address layout assumes that the board is configured with a base address offset of 0. Simply write the desired servo position value at the listed address to control the servo position. Note how the MSB and LSB registers are laid out. If 8-bit Control mode is adequate, then set the LSB to 0 and use only the MSB registers.*

daisy-chain configuration will receive the broadcast message, it's necessary to specify at address 0xF0 the destination base address of the board for which the message was intended. The selected board will then reply to these commands with the requested data.

The layout of the PWM registers is shown in Figure 5. The boards were designed to be daisy-chained so that the RS-232 receive line could be shared among several of them. Each board can have a unique base address from which all the register addresses are offset.

PROTOTYPE SETUP

In order to prototype the servo control board, I used a Spartan IIE development kit coupled with a hand-prototyped MCU board with the ATmega8515L. The two boards were linked by wire wrap, and the resulting prototype was put to the test-driving multiple servos. Photo 1 shows the prototype setup driving a custom-machined, multi-axis robotic pan/tilt unit.

One of the biggest prototyping challenges was keeping the wire lengths reasonably short. In the early debugging phase, the ALE line was too long and it generated spurious triggers that corrupted the FPGA registers. A compact PCB layout can solve a lot of these problems. The PWM timing resolution and accuracy figures were all verified on an oscilloscope.

PC SOFTWARE

None of the commercial programs provided 64 parallel sliders to drive the servo controller board and control the 32 coarse/fine servo position registers. Custom set-up commands were also required. As a result, I implemented a custom PC control software application with Microsoft Visual C++ 6.0 and MFC (see Photo 2, p. 48).

The blue bars in Photo 2 indicate the programmed servo travel limits for each channel. The top row of 32 sliders controls the MSB or coarse position of the associated servo channel. The bottom sliders control the respective LSB or fine position control. Several other menus enable you to set the speed, start positions, and other board configuration parameters. All these values can be sent to the target board and saved in the on-board EEPROM. The settings also can be read back from the board and displayed on the GUI.

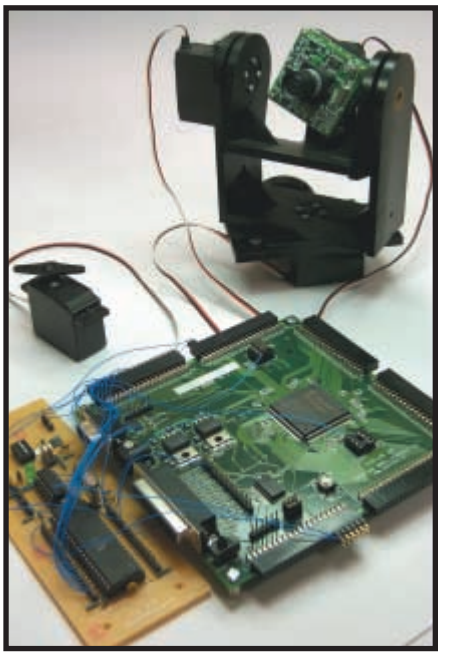
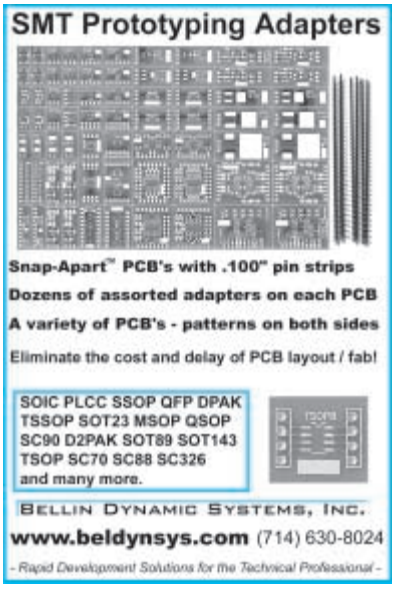


Photo 1—*My 32-channel digital RC servo controller circuit board prototype features a Spartan IIE development board along with a hand-prototyped MCU board. This circuit is driving a multi-axis robotic camera pan/tilt unit.*

APPLICATIONS

There are many interesting applications for the 32-channel RC servo controller. For example, insect and bipedal walker robots require a large number of servos to generate their

CCS Inc
CUSTOM COMPUTER SERVICES, INC.
USB DEVELOPMENT KIT
KIT INCLUDES EVERYTHING NEEDED, INCLUDING IN-CIRCUIT DEBUGGING & IN-CIRCUIT SERIAL PROGRAMMING, CCS C COMPILER, CABLES AND MORE!
FULLY ASSEMBLED DEMO BOARD WITH SCHEMATICS & EXERCISE BOOK OF READY-TO-USE EXAMPLES
USES A PIC16F877 WITH EXTERNAL FULL SPEED USB PERIPHERAL
EXAMPLES PROVIDED IN C, SHOWING A HID MOUSE, BULK MODE OSCILLOSCOPE & MORE!
HURRY!!
FOR A LIMITED TIME ONLY, RECEIVE A FREE MOUSE PAD & GROUND SHIPPING WHEN YOU USE CODE CC0505
OFFER EXPIRES: JULY 31ST 2005


SMT Prototyping Adapters
Snap-Apart™ PCB's with .100" pin strips
Dozens of assorted adapters on each PCB
A variety of PCB's - patterns on both sides
Eliminate the cost and delay of PCB layout / fab!
SOIC PLCC SSOP QFP DPAK
TSSOP SOT23 MSOP QSOP
SC90 D2PAK SOT89 SOT143
TSOP SC70 SC88 SC326
and many more.
BELLIN DYNAMIC SYSTEMS, INC.
www.beldynsys.com (714) 630-8024
- Rapid Development Solutions for the Technical Professional -

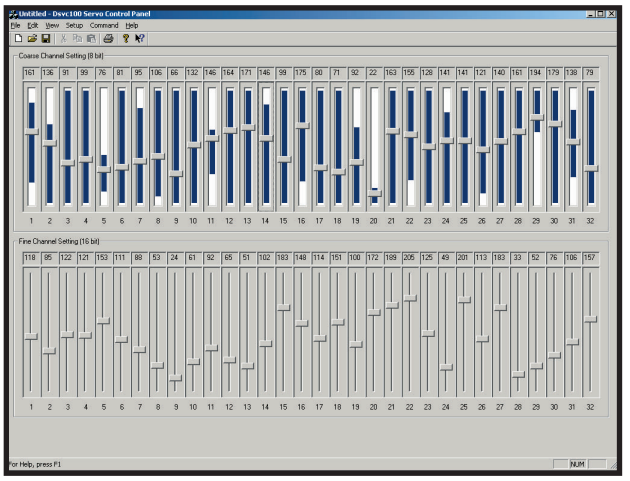


Photo 2—*The blue bars show the programmed travel limits of each axis. The top sliders are the coarse position control, and the bottom sliders are the corresponding fine position control. Other menus allow you to adjust the speed, start position, and other configurable parameters.*

walking motion.

Several university research labs currently specialize in nonholonomic robotic manipulators that have many degrees of freedom. An articulated snake with servo-driven sections is a good example of such a manipulator. Again, there's a requirement for controlling numerous servo channels.

The popular field of special effects can also benefit from multichannel servo control. Even the simplest full-featured animatronics creations typically require a large number of servos to bring them to life.

Daniel Rozin's wooden mirror project is another interesting application (http://fargo.itp.tsoa.nyu.edu/~danny/mirror.html). The artificial mirror consists of a large array of small wooden blocks actuated by 830 discrete servos. It can generate a pseudo image. Daniel used more than 100 serial servo controllers with eight PWM channels per controller. The same could be achieved with only 26 units.

SIMPLE SERVO CONTROL

Now you know how I built a 32-channel, high-accuracy, high-resolution RC servo controller with a simple serial interface. I ensured compatibility with the industry-standard Mini SSC-II control protocol. I even made a few improvements to the protocol. I also developed Windows-based PC control software from the ground up.

The entire process went smoothly and the controller tested well. The ATmega8515L microcontroller proved to be an excellent fit. Firmware development was extremely easy with the ImageCraft compiler. The on-chip-debugger proved extremely useful too. Combining the raw brute force of an FPGA with the intelligence of an MCU can yield end results that neither could realize easily on its own.

Eric Gagnon, M.A.Sc., P.Eng., has been hooked on electronics since the age of 12. He earned two degrees in electrical engineering from the University of Ottawa. Eric has more than 10 years of embedded design experience. He has worked on projects related to the International Space Station, industrial robotics, 3-D machine vision, and embedded video. Eric currently runs Digital Creation Labs, Inc. (www.digitalcreationlabs.com). You can contact him at egagnon@digitalcreationlabs.com.

PROJECT FILES

To download the code, go to ftp.circuitcellar.com/pub/Circuit_Cellar/2005/178.

RESOURCE

Insect and bipedal walkers
Lynxmotion
www.lynxmotion.com

SOURCES

ATmega8151L Microcontroller
Atmel Corp.
www.atmel.com

AVR C Compiler
ImageCraft
www.imagecraft.com

Spartan IIE development kit
Xilinx, Inc.
www.xilinx.com

VOLKSCOMPUTER!

PC Compatible Complete with Display and Keyboard for Industrial,Robotics,Security,Data Logging,Education,etc.

$27 VC1 (standard) OEM qty 1k

$95 eval kit (qty 1)

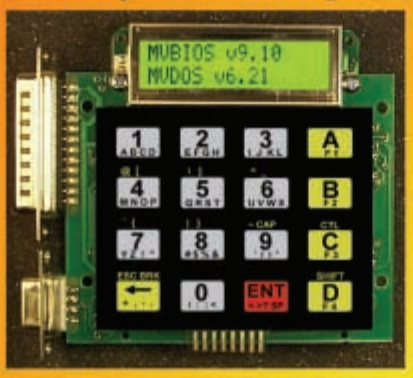
VC2 (more memory/ports) $42/$150

Parallel port (378h) drives PC printer or use as ttl I/O

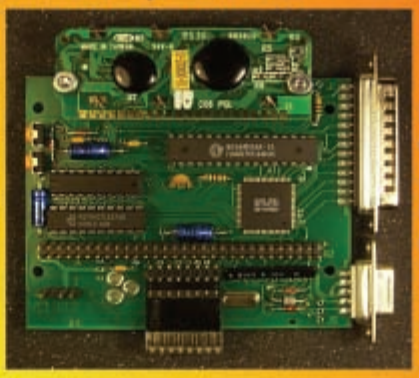
Tactile-click keys w/unique single action access to entire alphanumeric character set

Up to 4 serial lines can be configured for RS232 or multi-drop (3F8...2E8)

Ultra-sharp Super Twist II LCD with low power LED backlight



10 bit hi-speed ADC channels full scale adjustable 50mv-5v



2.75"x3.5" for easy mounting in small case or behind panels

On-board memory to 640K or to 1 Mbyt via bus (VC2)

Revolutionary microcoded CPU executes subset of PC Athlon/Pentium instructions so you debug on desktop then download to VolksComputer

ISA/104 bus for interfacing video, modem, hard drive, FLASH adapter cards

TRUE FRAME GRABBER

-Color & BW up to 640x480
-Serial/parallel:pic,avr,z80,pc
-Full speed: to 30 frames/sec
-Simultaneous composite out
-Use w/digital CMOS camera
-C, BASIC, Assembler source

oem(1k) **$27**

eval kit(1) **$95**

Add vision at low cost! Perfect for inspection, security,robotics. Full frame buffer unlike sx, pci,usb type. Industry std PC104 form factor.

SINGLE CHIP COMPUTER $7

eval kit oem (1k) $1.99

-ZERO external components
-Built-in BASIC / Assembly
-RS232 program download
-1K flash,64ee,3irq,2timers
-15 I/O bits, A/D comparator
-20mips,faster than pic/8051
-20 pin DIP part #MV1200

NEW! 8K SUPER CHIP

40x the BASIC program space!
-32 i/o, 12 irq, 3 timers, Bus
-8K flash, 512ee, 512nvram
-Watchdog with internal osc.
-40 pin DIP part #MV8515
oem(1k) $7.10 eval(1) $25.00

OPTIONS:
-SERIAL EE ADAPT.
-32 PIN ZIF SOCKET
-PLCC ADAPTER

$95 PRINTER PORT PROGRAMMER

FLASH, EE, NVRAM, EPROM to 8meg. Adapters for micros. Super small! Stand alone or drive bay mounting kits available. WINDOWS & DOS software. Script files to simplify operation. High yield and faster program than others. 'Gang Bang' option.

SERIAL MINI-TERMINAL

RS232 terminal for Stamp, PC, Z80, AVR etc.
-super low-current, powers from serial line
-LED backlit LCD, visible in all conditions
-115.2kbps, DB9 conn, simple commands
-specify 20 customizable or 16 tactile keys
eval(1) $75,oem(1k) $21.30,w/BASIC cpu $27

SERIAL TO ATA STAMP DRIVE

Read / Write PC compatible hard disk, PCMCIA, & Compact Flash. RS232 to ATA drive adapter for Stamp, 8051, AVR, PIC, Z80, x86 ANY controller, big or small:

- **-up to 4 gigabyte capacity**
- **-low power operation 5v 2ma**
- **-simple software commands**
- **-baud rates up to 115.2kbps**
- **-$14 IDE & $21 ISA/104 versions**

oem(1k) **$27**

eval kit(1) **$95**

WWW.STAR.NET/PEOPLE/~MVS
Alternate: WWW.GEOCITIES.COM/MEGAMVS

MVS Box 803
Nashua,NH 03060
(508) 792 9507

.MVS

5yr Limited Warranty
Free Shipping
Mon-Fri 10-6 EST

Serving the Embedded Community Since 1979!

Everywhere you imagine. RENESAS

MCU Solutions Matched To Your Design Imagination

RENESAS IS THE #1 MCU SUPPLIER IN THE WORLD. **www.renesas.com**

M16C Advantages

- Advanced 16-bit and 32-bit CISC cores that perform at up to 32 Dhrystone 1.1 MIPS
- Banked-register architecture optimized for fast context switching
- Wide range of high-performance flash memory: 8 KB to 512 KB
- Best-in-class EMI/EMS noise immunity
- Designed to be pin- and function-compatible between families and packages
- Extensive peripheral integration: everything from POR and LVD to CRC counters

Low-cost Renesas Toolchain

- Fully integrated suite of hardware and software tools
- Highly-optimized C compiler and assembler — FREE (64K limited version): Download latest version from web (www.renesas.com/skp)
- Low-cost Starter Kit (target board, USB debugger, complete toolchain), starting at $49
- Easy online access to application notes, example code, software development tools

Get your product to market faster using Renesas Interactive!

Renesas INTERACTIVE

Evaluate, test, and learn about Renesas products in an interactive online environment.

www.renesasinteractive.com

SKP	Device	C Compiler, Assembler, IDE	Cost
SKP8CMINI13	R5F21134FP	FREE (64K limited version)	$49
SKP8CMINI17	R5F21174FP		$49
SKP16C26A	M30260F8AGP		$49
SKP16C28	M30280FAHP		$49
SKP16C62P	M30626FHPFP		$49
SKP32C83	M30835FJGP		$119
SKP32C84	M30845FJGP		$49

Special pricing is in US dollars from Renesas authorized North American distributors only, and is subject to change.

Popular Devices

Series/ Group	Part Number	CPU (Bits)	Package	Freq. (MHz)	Dhrystone 1.1 MIPS (Max)	Program Flash (KB)	Data Flash (KB) (10K Rewrites)	SRAM (Bytes)	Serial I/O Channels	I²C	I/O	A/D 10-bit	Ring Osc.	LVD POR	Timers	Price *
R8C/17	R5F21172SP	16	20 Pin SSOP	20	9	8	2	512	1	1	15	4	X	X	3+WDT	$1.90*
R8C/13	R5F21134FP	16	32 Pin LQFP	20	9	16	4	1K	2	–	24	12	X	X	4+WDT	$2.38*
M16C/26A	M30260F8AGP	16	48 Pin LQFP	20	13	64	4	2K	3	1	39	12	X	X	8+WDT	$4.76*
M16C/28	M30281FAHP	16	64 Pin LQFP	20	13	96	4	8K	4	1	55	13	X	X	9+WDT	$5.90*
M16C/62P	M30624FGPGP	16	100 Pin QFP	24	16	256	4	20K	5	3	87	26	X	X	11+WDT	$8.25*
M16C/62P	M30626FHPGP	16	100 Pin QFP	24	16	384	4	31K	5	3	87	26	X	X	11+WDT	$10.05*
M32C/84	M30845FJGP	16	144 Pin LQFP	32	32	512	–	24K	5	5	124	34	X	X	11+WDT	$16.88*

* Suggested 10,000 piece price. Please contact your local Renesas distributor for volume pricing.

Three-Axis Stepper Motor Controller (Part 2)

Software Implementation

Now that you can design a three-axis stepper motor driver, it's time to learn how to control it. You can control the system locally with a keypad and remotely via an RS-232 serial interface.

Last month we described our stepper motor controller design. Now we'll explain how to implement software that will enable you to control the motors locally via a front panel keypad and remotely via an RS-232 serial interface.

We built a general-purpose controller. Different applications have different motion requirements (e.g., constant speed motion, programmable speed, torque, step size, rotary movement, etc.). If you want your hardware to satisfy every application, it must be flexible and highly configurable. A typical example involves driving the motor with and without a gearbox. In order to meet both requirements, you must specify the gear ratio as a parameter in the firmware and preserve it. To control the system remotely, the available options must be configurable through the firmware or application software running on a PC.

The hardware already provides numerous customization options (e.g., motor current, speed, Half/Full Step mode operation, independent on/off control, etc.) because we used separate hardware for each motor. In order to execute the different options at runtime, the firmware must be flexible too. The firmware must be able to handle multiple tasks (e.g., acknowledging serial commands while driving three motors simultaneously) at the same time via one CPU. Fortunately, the Philips P89C51RD2 microcontroller has 64 KB of downloadable on-chip code flash memory. This memory, along with 1 KB of on-chip data RAM, provides enough room to accommodate complex firmware code without expanding the chip further and preserving the precious and powerful I/O port pins for motor control. This is probably the reason why we could accommodate a third motor axis into our initial conceptual design for two motor axes.

CONFIGURABLE FIRMWARE

Different applications require different parameters. After we determined all the possible parameters associated with motor control, we assembled it into a data structure (see Listing 1, p. 52). Choosing the right algorithms is of key importance when you're working with an embedded system characterized by a limited user interface, physical size constraints, and strict limits on resources (memory, power consumption, etc.).

We declared a data type (`MOTOR`) and a single-dimension array variable (`MotorData[]`) to statically allocate memory space for three motors. Although the Keil 8051 C compiler supports dynamic memory allocation, you shouldn't use it for embedded system programming. Using an array of structures has another advantage. You can easily extend it by increasing the array size if you need four or more stepper motor axes in your application.

We used the `bdata` modifier to declare the `Motor[]` array variable in the bit-addressable space in the on-chip 8051 data memory. The variable stores the current state of each motor-control bit (e.g., clock pin, motor direction, and limit switches). The MCS-51 architecture enables you to store 128 bit variables. The Keil C compiler makes it easy to assign bit variables to the bit space. Because of the limited amount of space, Listing 1 shows only the definitions for the X motor. There are similar definitions for the other motors.

Each motor's configuration information is saved in the 128-byte EEPROM. This enables you to load the parameters at power-on reset and initialize the `MotorData` structure. Relocatable definitions were made for each motor axis (see Listing 1). If you make a change in the `MOTOR` data structure, you must make the same change in the definitions. We used the `GetMotorDataE2P()` data structure to load the data from the EEPROM.

Because the elements inside an array occupy adjacent bytes, we had to load a block of data equal to the size of the `structure _motor` data structure. The `MOTOR` data type was defined as the union of `structure _motor` and a byte array of the same size, so data transferred from the EEPROM to the `MotorData` variable was greatly simplified. Moreover, we didn't need to bother with the data type of the individual structure elements because it was already taken care of by the union.

You can change few parameters when you're operating the driver. If you need to store parameters immediately after the change to the EEPROM, you can use another definition with overlapping storage (union declaration) for the specific data type (e.g., a float variable with the byte array size of `sizeof(float)`). You can then insert the float value into the float variable.

You'll get individual data bytes by accessing the byte array. You can declare a similar data structure for other intrinsic data types. This relieves you from having to manage the bytes while moving individual parameters between the EEPROM and data RAM. You also save on code even though the P89C51RD2 provides plenty of code memory.

USER INTERFACE

You can control the motors either locally via a keypad or remotely via an RS-232 serial interface. The driver's status is updated periodically on the local 4 × 20 LCD. Additional code is required to communicate with the AT89C2051 slave device (see Listing 2). A bit-banging serial UART is used for this purpose. The command response protocol is necessary to avoid data loss when the P89C51RD2 is paying attention to another important task, and it can't receive the serial data in the background. A reserved port bit notifies the host that you hit a key. The host then conveniently sends a command to read the key code.

The bit-banging UART code (`putc` and `getc` functions) is implemented in assembly language and linked to the C code. The `AccessKBLCDSlave` function sends a command and waits for a response from the slave (see Listing 2). This notifies the slave to get ready for additional communication. The host may send more data (typically the `GOTOXY` and `Str2LCD` functions) and close the communication.

There are many configurable parameters. The Menu key invokes the configuration menu. The parameters for each axis are separate but identical; therefore, you can use the same string of menu items to set identical parameters, provided you can change the axis while the menu is displayed.

The `MotorData` structure declaration comes to the rescue. You need to provide only the array index to match the currently selected axis. This enables you to reuse the menu item handler functions for all of the axes. Each menu item string is limited to a maximum of 20 characters. The parameter values are displayed or the new values are entered on another line on the LCD. You must write the menu handler function corresponding to each menu item. We developed a library menu for our users. You may download the code for our menu from the *Circuit Cellar* ftp site.

We prepared a comprehensive command set to remotely operate the driver (see Table 1, p. 54). This enabled our users to automate the experiments that included our stepper motor drivers. The Standard Commands for Programmable Instrumentation (SCPI) defines a standard set of commands

Listing 1—*Using a data structure greatly simplifies feature implementation. The configuration management code fragment effectively uses this data structure to access the configuration motor parameters.*

```
struct _motor {
  long Pos;             // Motor current position
  int Speed;            // 1 - High, 2 - Medium, 3-Low
  char Movement;        // 1 - coarse, 2 – fine
  unsigned CoarseSteps; // Number of steps to move in coarse movement
                        // mode
  char FineSteps;       // Number of steps to move in fine movement mode
  char StepMode;        // 1 - half step, 2- full step
  char MoveMode;        // 1 - steps, 2- linear, 3 – angular
  long MaxTravelLength; // Either in millimeters or angle or steps
  float ScrewPitch;     // steps per millimeter
  intGearRatio;         // e.g., 1:1000
  char CheckLimit;      // 1 - check, 0 - Ignore limit switches
  char HomeSw;          // Search Home 1 - Home at LimSw1,
                        // 2 - home at LimSw2
};
typedef union {
  struct _motor     mot;
  unsigned char motbytes[sizeof (struct _motor)];
}MOTOR;
MOTOR MotorData[3];     // Declare three variables for X, Y, and Z
motors
bdata unsigned char Motor[3];    // Current status and settings for motors
sbit XDir        = Motor[0]^0;  // Direction
sbit XEn         = Motor[0]^1;  // Enable
sbit XLim1       = Motor[0]^2;  // Lim1=0 if lim1 reached
sbit XLim2       = Motor[0]^3; // Lim1=0 if lim2 reached
sbit XEnLimSw1   = Motor[0]^4;  // If enabled, check limit switch 1
sbit XEnLimSw2   = Motor[0]^5;  // If enabled, check limit switch 2
sbit XStepMode   = Motor[0]^6;  // Half step = 1, full step = 0
#define LOC_MOTORDATA            0
#define XMOTORDATA_BASE (LOC_MOTORDATA + (XMOTOR-1)*MOTOR DATA_   SIZE)
#define YMOTORDATA_BASE (LOC_MOTORDATA + (YMOTOR-1)*MOTORDATA_SIZE)
#define ZMOTORDATA_BASE (LOC_MOTORDATA +  (ZMOTOR-1)*MOTORDATA_SIZE)
#define LOC_POS 0                                          // long
#define LOC_SPEEDLOC_POS + sizeof(long)                    // int
#define LOC_MOVEMENT LOC_SPEED + sizeof(int)               // char
#define LOC_COARSE_STEPS LOC_MOVEMENT + sizeof(char)   // unsigned
#define LOC_FINE_STEPSLOC_COARSE_STEPS + sizeof(unsigned)  // char
#define LOC_STEPMODE    LOC_FINE_STEPS + sizeof(char)   // char
#define LOC_MOVEMODE    LOC_STEPMODE + sizeof(char)     // char
#define LOC_MAX_TRVEL   LOC_MOVEMODE + sizeof(char)     // long
#define LOC_SCREWPITCH  LOC_MAX_TRVEL + sizeof(long)    // float
#define LOC_CHECKLIMIT  LOC_SCREWPITCH + sizeof(float)  // char
#define LOC_HOMESWLOC_CHECKLIMIT + sizeof(char)         // char
#define MOTORDATA_SIZE (2*sizeof(long)+ 6*sizeof(char)+ 2*sizeof(int) +
                sizeof(float))
void GetMotorDataE2P (char motor)
{
unsigned char loc = LOC_MOTORDATA + (motor-1)*MOTORDATA_SIZE;
int i;
        for (i=0; i < sizeof(MOTOR); i++) {
             MotorData[motor-1].motbytes[i] = E2PReadByte(loc+i);
  }
}
typedef union {
        float f;
        unsigned char b[sizeof(float)];
}uFloatBytes;
```

for controlling programmable measurement and test devices in instrumentation systems. We derived most of the commands from the SCPI Consortium's guidelines. Adhering to the standard helped us use the instrumentation effectively. It also enables us to reuse the SCPI programs already written for similar instruments. Most standard instruments, such as oscilloscopes and multimeters, use the SCPI for their remote interfaces.

The P89C51RD2 microcontroller has an on-chip UART for serial communication. Interrupt-driven code receives the remote commands in the background while the driver works the motors. A circular buffer implementation accumulates the command characters. The character parser function examines each received character, transmits an echo, announces the detection of query commands, handles backspace character for line editing, and, finally, assembles a command string. Note that the command parser code separates the command and arguments after it's validated. This is accomplished with a string look-up table. Another routine executes the command.

Listing 2—*A simple code fragment generates master/slave interaction between the host and keypad display slave controller. This is a typical command code, although more commands are actually implemented.*

```
#include <Reg51.h>
bit fRecvErr1;            // Recv error on slave
extern unsigned char getc (void);
extern void putc ( unsigned char ch);
#define SLAVE_STATUS 0    // Read keyboard status
#define GET_KBSTATUS 1    // Read keyboard status
#define GET_KEY      2    // Read last key pressed, 0xFF -> no key
#define CLRLCD       3    // Clear LCD
#define DSPCHR       4    // Display one char at current pos
#define GOTOXY       5    // Set cursor position
#define DSPSTR       6    // Display string at current loc
unsigned char KB_Status, KeyCode;
unsigned char Slave_Status;
char InOutBuf[20];        // 20-byte comm buffer
void AccessKBLCDSlave (char cmd)
{
char c;
    putc(cmd);
    c = getc();
    switch (cmd) {
         case GET_KEY:
                    KeyCode = c;
         break;
         case GOTOXY:
              putc (InOutBuf[0]);          // x position
              putc (InOutBuf[1]);          // y position
         break;
         case DSPSTR:
         {
         char i=1;
              putc(c=InOutBuf[0]);         // strlen
              while(c—) putc(InOutBuf[i++]); // Send string
         }
         break;
         case SLAVE_STATUS:                // Return slave status
              Slave_Status =c;
         break;
    }
}
void Str2Lcd (char *s)
{
char i=1;
    while (*s) InOutBuf[i++] = *s++;
    InOutBuf[0] = i-1;                     // Compute strlen
    AccessKBLCDSlave (DSPSTR);
}
char GetKey (void)
{
    AccessKBLCDSlave(GET_KEY);
    return (KeyCode);
}
void GotoXY (char x, char y)
{
    InOutBuf[0] = x;
    InOutBuf[1] = y;
    AccessKBLCDSlave(GOTOXY);
}
```

TASK MANAGEMENT

If your software will be a part of a product shipped in large volumes to remote locations, keep in mind that it will be extremely expensive to update if errors are discovered. A common error involves overlooking event combinations that occur when an application runs in an intended environment.

Your firmware should be organized in a modular, hierarchical fashion. Each independent task should be in a separate module. This will help with testing, maintenance, and debugging.

By treating each motor axis independently, we made it easier to write the firmware. We needed to be able to handle multiple independent parallel tasks (e.g., controlling the user interface from a keypad or remote PC), system configuration management tasks, and individual motor axis control tasks. We also needed to be able to synchronize them if required.

A real-time operating system (RTOS) enables you to create, synchronize, assign, and prioritize various tasks. Another alternative is to use a state chart or state machine event-driven programming technique to accomplish these objectives in a timely manner. Both approaches enable multitasking.

An RTOS framework provides tight timing and deterministic performance. You need to select an RTOS to suit your development tools and microcontroller. If you're new to real-time programming, you must first learn the RTOS and programming techniques. Keep in mind that when

SN	Command	Arguments	Description
1	MOVR	Motor, number of steps	Move specified motor relative to current position by specified number of steps
	MORA	Motor, absolute position	Move specified motor absolute position
2	IDN?	–	Identify yourself query command
3	RST	–	Perform a warm reset
4	LERR?	Optional motor	Get error status of last command
5	HOME	Motor	Get motor to its home position
6	HOME?	Motor	Is motor at home position?
7	PICH	Motor, value	Set the pitch for the motor
8	GEAR	Motor, value	Set the gear ratio for the motor
9	STAT	Motor	Get status of the motor
10	CURR	Motor	Set the winding current for the motor
11	MODE	Motor, value	Set the operating mode (linear, rotational, or half/full step) for the motor
12	SYST	Mode/status	Set the system to remote or local mode of operation or request the current mode
13	CONF	Motor, parameter	Set the configuration information for the motor
14	CONF?	Motor, parameter	Get the configuration information for the motor
15	SPED	Motor, value in RPM	Set the motor speed in RPM
16	STOP	Motor	Stop the specified motor
17	UNIT	Motor, value	Set the operational unit (mm, cm, or steps) for the motor
18	POSI	Motor	Get the current position of the motor in the current motion unit

Command	Description
MOVR:X:100	Move X motor relative to current position by 100 steps clockwise
SPED:Y:100	Set Y motor speed to 100 RPM
UNIT:X:MM	Set X motor motion unit in mm
CONF:X:SPED:150	Set X motor speed to 150 RPM and save to EEPROM
CONF:Y:SPED?	Get Y motor speed value (in RPM) from EEPROM
MODE:Y:HALF	Set the Y motor in half-step operation
SYST:MODE?	Get current operating mode for the drive?
*IDN?	Report identification string for the system

Table 1—*We implemented a comprehensive command set for remote motor control. It's based on SCPI commands. A few commands have subcommands with a colon character acting as a separator.*

using an RTOS, you'll sometimes have to pay a per-unit royalty fee. Coding a well-defined state machine, however, is relatively simple. The stepper motor driver doesn't demand critical timing. Moreover, the programmable counter array (PCA) channels that we used to time the motors took care of the required timing in the background ISR.

We assigned one PCA channel to each motor control. Because PCA hardware can do its job without CPU intervention, our motors ran in the background while the CPU was serving the RS-232, keypad, EEPROM, and other entities. You can easily convert a linear C program into well-behaved state machine code.

STATE MACHINE

A state machine has a finite set of states and a collection of transitions. The states represent the distinct stages of computation, and the transitions represent the way in which the computation process moves from one stage to another.

In terms of developing embedded software, state machines are primarily used to capture the flow of the computation. Data manipulation, such as arithmetic and data structure, is handled separately. You need to model your embedded application as a sequence of transitions between states. In each state, events (inputs) will change the state and possibly generate an action (output). The state machine code is thus a purely reactive engine core, as opposed to the RTOS, which is complete framework for developing multitasking software. For applications based on tiny single-chip microcontrollers with restricted memory, you can't afford operating system overhead. But you can handle the multitasking by modeling the application as one or more state machines.

You can model the simple embedded computation needed for controlling a stepper motor as a state machine (see Figure 1). The state machine has states corresponding to the actual stages a motor goes through during operation. The S_0 state initializes the state machine and data associated with the motor (e.g., its default speed, direction, and Stepping mode). After initialization, the state isn't revisited unless you reset the driver.

Each state machine maintains a state variable that indicates the machine's current state. The state machine code is simply a C language switch statement construct (see Listing 3, p. 59). The state variable selects the case. Each state numbered sequentially is declared within the case value branch.

The state machine code assumes that events or inputs are monitored in another section of the code. After a relevant event occurs, the state machine code is called. Assuming that events (whether variable or flag) are accessible to the state machine code, it would branch to the current state, examine the relevant events, and manipulate the state variable to point to the next state as per the state chart diagram. In addition, it calls data manipulation routines, such as `RunMotor()` and `StopMotor()`, to generate the appropriate signals for physical actions.

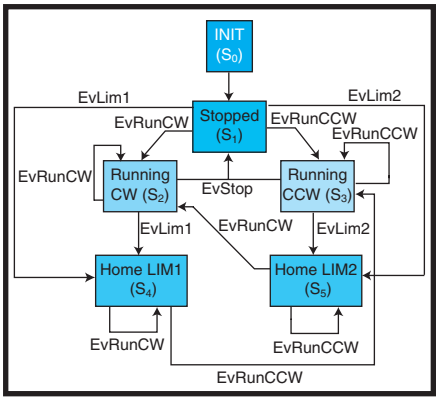


Figure 1—*We used this state machine chart model for stepper motor control. The arrows indicate the transitions between the states that result after events. Sometimes actions are taken after a transition.*

picc enterprise edition

300+ supported chips
3 reliable compilers
1 simple solution

HI-TECH Software now brings you three award-winning compilers in one with the new PICC Enterprise Edition.

upgrade offers available

HI-TECH SOFTWARE

HI-TECH Software's **PICC Enterprise Edition** is the only compiler package available with comprehensive support for all 10/12/14/16/17/18 series PICs and also the new dsPIC processors. PICC Enterprise Edition is the complete package that unites our successful PICC, PICC-18 and dsPICC compilers.

With the choice of over 300 different PIC processors, deciding which one to use is a challenge. Even when you do decide, changing project requirements during development often call for a larger or more sophisticated chip. A change in processor can also mean buying a new compiler...until now!

With HI-TECH Software's PICC Enterprise Edition, there is finally a single product that supports all of Microchip's PICmicro processors. Not only does this simplify which compiler to choose, it also eases migration between processors by having a familiar tool-set.

PICC

Our most popular genuine ANSI C compiler, PICC is a high-performance C compiler for the Microchip PICmicro 10/12/14/16/17 series of microcontrollers. PICC supports all standard data types including 24 and 32 bit IEEE standard floating point.

This industrial-strength compiler makes full use of specific PIC features and using an intelligent optimizer, can generate high-quality code easily rivaling hand-written assembler.

HI-TECH Software's PICC compiler can be fully integrated with MPLAB, or can be used directly from a Makefile or command line.

PICC-18

HI-TECH Software's PICC-18 compiler is a proven C compiler for the Microchip PICmicro PIC18 family of microcontrollers. Delivering reliable efficient code that's as good as or better than hand-written assembler code.

Tightly tuned to the PIC18 architecture, PICC-18 allows firmware development in a fraction of the time required for conventional assembly language.

HI-TECH Software's PICC-18 implements full ISO/ANSI C (with the exception of recursion) and integrates with the HI-TIDE IDE or with MPLAB.

ds PICC

HI-TECH Software's dsPICC is the outstanding choice for developers on Microchip's newest family of PICmicro devices.

Built with the same technology as our highly successful PICC and PICC-18 compilers, dsPICC promises the same reliability and code efficiency to which you have become accustomed.

Full ISO/ANSI C is supported, including recursive function calls.

HI-TECH Software's dsPICC integrates seamlessly with MPLAB and HI-TIDE, or can be run directly from the command line.

Recognized Microchip Third-Party Tool Provider

HI-TECH Software proudly supports the Microchip brand with industrial-strength software development tools and C compilers

HI-TECH Software LLC 6600 Silacci Way Gilroy, CA 95020 USA
Ph: 800 735 5715 **Web:** http://www.htsoft.com/

THE LEADING INDUSTRY EVENT FOR SENSING TECHNOLOGIES

sensors expo & conference

June 6 - 9, 2005 • Chicago, Illinois

The largest Expo and Conference in North America focusing on sensors and sensor integrated technologies, serving the product resource and information needs of engineers, R&D, corporate executives, system architects and systems integrators.

Three Full Day Symposia:

Automotive Sensing • Medical Sensing
Wireless Sensing

A comprehensive conference program featuring high visibility keynotes and over 100 speakers.

Conference Tracks:

Intelligent Systems • Technology & Design
Industry Applications • Business Perspectives

Leading exhibitors featuring sensors, instrumentation and controls, and integrated technologies.

Register Today! www.sensorsexpo.com or 800-331-5706
(outside the U.S. and Canada, please call 218-723-9130

To Exhibit, Please Contact:

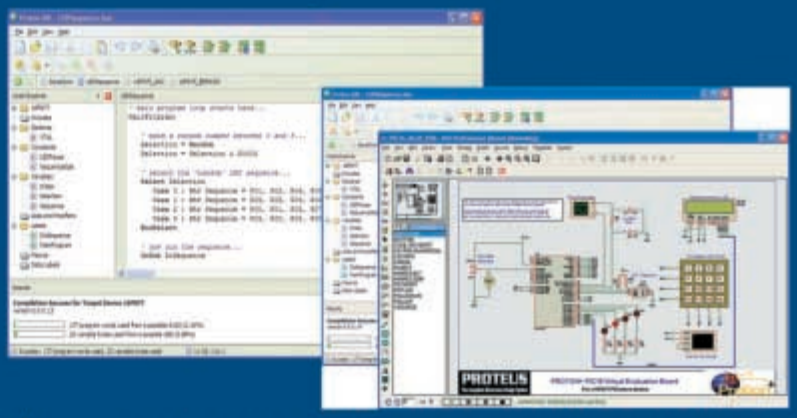
Jeanne Duval (companies A-M) • 603-924-5409 • jduval@advanstar.com
Cathy Walters (companies N-Z) • 516-771-6868 • cwalters@advanstar.com

Produced By: ADVANSTAR TECHNOLOGY GROUP

To Register, Please Use Source Code: 301M

PIC Programming Made Easy!

Proton+ PICBASIC Development Suite



Next Generation IDE

Proton IDE is a professional and powerful visual Integrated Development Environment (IDE) which has been designed specifically for the Proton Plus compiler. Proton IDE accelerates product development in a comfortable user environment without compromising performance, flexibility or control.

- *Code Explorer*
- *Compiler Results*
- *Programmer Integration*
- *Integrated Bootloader*
- *Real Time Simulation Support*
- *Serial Communicator*
- *Online Updating*
- *Plugin Architecture*

INTRODUCTORY OFFER! Save $10 LIMITED TIME

SAVE TIME WITH END TO END INTEGRATIONS

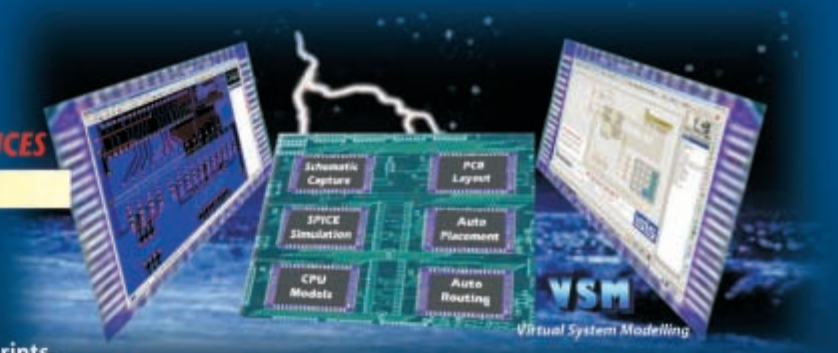
NEW IDE - New IDE makes development using Proton+ even faster and more intuitive!
COMPILER - The popular Proton+ compiler has enhanced support for I²C, SPI, Dallas 1-wire bus, RS232, X10, Compact Flash Memory Cards and USB.
VIRTUAL SIMULATION - Simulate your project in RealTime using the integrate Proteus Virtual PIC Boards.

Visit www.r4systems.com to see our latest projects using **Proton+**

PROTEUS

The Complete Electronics Design System

EASY TO USE CAD TOOLS AT FANTASTIC PRICES



Schematic and PCB Layout

- Powerful and flexible schematic capture.
- Auto-component placement.
- Rip-up and Retry PCB routing.
- Polygonal gridless ground planes.
- Library of over 8000 schematic and 1000 PCB foot prints.
- Bill of materials, DRC reports and more.

Mixed Mode SPICE Circuit Simulation

- Berkeley SPICE3F5 simulator with custom extensions for true mixed mode and interactive simulation.
- Six virtual instruments and 14 graph based analysis types.
- 6,000 models including TTL, CMOS and PLD digital parts.
- Fully compatible with manufacturers' SPICE models.

Proteus VSM - Co-simulation and debugging for popular Micro-Controllers

- Supports PIC16 & PIC12, AVR, 8051, HC11 and ARM micro-controllers. Latest version includes 40 new PIC18's.
- Co-simulate target firmware with your hardware design.
- Includes interactive peripheral models for LED and LCD displays, switches, keypads, virtual terminal and much more.
- Provides source level debugging for popular compilers and assemblers from HiTech PICC, Crownhill, IAR, Keil and others.

PCB AutoRouting

- Proteus PCB design includes an interface to the Electra Gridless autorouter.

"For the cost of the software compared to the productivity gains, I consider Proteus to be pivotal in the commercial viability of my company and by far represents the best value for money of anything Tempus possesses."

ROB YOUNGS, Tempus Consulting

FREE DOWNLOADABLE DEMO

Save Time. Save Money.

Proteus Starter Kit – $199

Complete Systems from $449

"This is clearly superior in every respect."

R4 SYSTEMS INC.

www.r4systems.com
Tel: 905•898•0665 info@r4systems.com

Remember when...

...electronics stores were stocked to the rafters

with every component imaginable and there was ALWAYS

a knowledgeable person there to help you?

Think those days are gone?

Well they're not!

You just need Jameco!

Call for a FREE 256-page catalog today.

JAMECO® ELECTRONICS

Great Products. Awesome Prices.

1.800.831.4242

We have the electronic components and knowledgeable Technical Support people who can help **bring back the fun in your electronics hobbies, inventions, and DIY projects.**

Be sure to visit our new website at

www.Jameco.com/ccj

Jameco Electronics

—where electronics is still fun!

The motor state machine code doesn't include code for managing the PCA. Management is carried out by data manipulation functions such as `SetMotorSpeed()`. Listing 3 shows how cases for the S_0, S_1, S_2, and S_4 states are handled. The code for S_3 and S_5 is identical to the code for the S_2 and S_4 state handlers; the only differences are the events handled (see Figure 1). The code controls all three motor axes. The appropriate state variable and events are selected by the `motor` variable, which is passed to the state machine as an argument. You must periodically call the state machine from an endless loop for each motor to properly control each axis. You can reuse the code to implement identical state machines.

Listing 3—*The reusable state machine represents a motor. Placing a call to this state machine with the motor axis as an argument invokes the state machine to control that motor axis. You must generate various events to trigger the state transitions.*

```
// The motor can have one of the following states
#define   MOTORSM_S0  0                     // S0 - Initiate the motor
#define   MOTORSM_S1  1                          // S1 – Stopped
#define   MOTORSM_S2  2                          // S2 - MOVING_CW
#define   MOTORSM_S3  3                          // S3 - MOVING_CCW
#define   MOTORSM_S4  4                          // S4 - HOME_LIMSW1
#define   MOTORSM_S5  5                          // S5 - HOME_LIMSW2
struct _motevents {
    char EvRunCW, EvRunCCW, EvStop, EvLim1, EvLim2;
} MotorEvents[3];
char MotorSM_State[3];
void MotorSM(char motor)
{
    switch (MotorSM_State[motor]) {
          case MOTORSM_S0:
          {
              MotorEvents[motor].EvRunCW =0;
              MotorEvents[motor].EvRunCCW =0;
              MotorEvents[motor].EvStop =0;
              MotorEvents[motor].EvRunCW =0;
              MotorEvents[motor].EvLim1 =0;
              MotorEvents[motor].EvLim2 =0;
              MotorSM_State[motor] = MOTORSM_S1;
          }
          break;
          case MOTORSM_S1:                       // Motor is stopped
          {
              MotorEvents[motor].EvStop =0;      // Reset the event
              if (MotorIsRunning(motor)) StopMotor(motor);
              if (MotorEvents[motor].EvRunCW) {
                  MotorEvents[motor].EvRunCW = 0; // Reset event
                  MotorSM_State[motor] = MOTORSM_S2;
                  SetMotorDirection(motor, DIR_CW);
                  RunMotor(motor);
              }
              if (MotorEvents[motor].EvRunCCW) {
                  MotorEvents[motor].EvRunCCW = 0; // Reset event
                  MotorSM_State[motor] = MOTORSM_S3;
                  SetMotorDirection(motor, DIR_CW);
                  RunMotor(motor);
              }
          }
          break;
          case MOTORSM_S2:         // Motor is running in CW direction
          {
              if (MotorEvents[motor].EvLim1) {
                  StopMotor(motor);
                  MotorSM_State[motor] = MOTORSM_S4;
              }
          }
          break;
          case MOTORSM_S4:                       // Motor is at home1
          {
              MotorEvents[motor].EvLim1 = 0;     // Reset event
          if (MotorEvents[motor].EvRunCCW) {
              MotorSM_State[motor] = MOTORSM_S3;
              SetMotorDirection(motor, DIR_CCW);
              RunMotor(motor);
          }
       }
       break;
   }
}
```

You can generate events for the motor state machine from a number of sources. For instance, a keypad handling code may trigger the `EvRunCW` event for an x-axis motor when you press a key on the front panel. The PCA ISR may trigger the `EvStop` event for a y-axis motor after reaching the terminal count requested by a `MOVR:Y:200` remote command. You must identify different tasks that need to run concurrently and model them as the state machines. A state machine can call another state machine as well. This way you can create mutually exclusive access to the shared resources.

As for the motor driver, if you want to disable the keypad while you're operating the driver in Remote mode, you need to declare the keypad handler and remote command handler as two different states of a single state machine. A single state machine obviously can't execute two states simultaneously.

MOTOR CONTROL ROUTINES

Now let's focus on programming the PCA hardware to control the motor. You can use the High Speed Output (HSO) mode of this hardware resource to move the motor by toggling the port pin in the hardware that provides the clock signal to the motor sequencer IC (L297).

While we were designing the driver hardware, we decided to allocate one 8-bit I/O port per motor. Because all five PCA modules were located on the low port 1 bits, we needed to reserve port 1 PCA bits for driving the L297's clock pin. The stepper motors wouldn't respond at higher speeds (the maximum clock frequency required wouldn't exceed 1 kHz or even less). So, instead of using the HSO mode, we used the PCA's Software Timer

mode and toggled the clock pin inside the PCA interrupt. This enabled us to use the PCA module to drive a motor without requiring us to use the module's dedicated hardware pin to drive the motor clock signal. The only difference between HSO mode and Software Timer mode is that the latter doesn't automatically toggle the module's pin.

Listing 4 is the PCA module0's hardware initialization code. The module is programmed to drive the X motor in Software Timer mode. The CCF0 flag indicates that the interrupt has been caused by module0. All PCA modules share a single interrupt vector location at 0033H, which is the physical code memory address. You must declare a jump instruction at this location to hook up the ISR. You can do so using a special interrupt keyword for the Keil C compiler. Inside the PCA ISR, you must toggle the corresponding clock pin to generate the square wave. You also must reload the module's compare and capture registers (a pair of two 8-bit registers, CCAPOH and CCAPOL) to get another interrupt corresponding to the motor speed.

The ISR also maintains one `MterminalCount[]` variable to keep track of the number of steps taken (see Listing 4). The ECCFn bit (of the CCAPMn Compare/Capture mode register) enables an interrupt when a module's event flag is set. The bit clears after a motor's terminal count is reached. The `RunMotor` and `StopMotor` routines illustrate this. The PCA ISR also generates an `EvStop` event, which signifies that the motor has stopped after reaching its desired position. You can call these routines from other code modules to stop or run a corresponding motor.

The Keil C compiler toolchain has an extremely powerful built-in simulator for each device supported in the device's database. You can write a program in interpreted C language to control the simulated device's various external inputs and timing. Moreover, the environment simulates the aspects of all of the on-chip peripherals. This will help you develop your code during the software development phase without the actual hardware and associated laboratory instruments (e.g., CRO and function generator).

CONTROL PROGRAM

We developed a control program in LabVIEW to control the driver from a PC (see Photo 1). Although the application controls laser lab instruments, you can use it as a guide for your own project. The application was developed

Listing 4—*The motor control PCA routine enables background motor driving. In PCA Software Timer mode, you can generate a square wave, which can be used to drive the motor along with the L297 sequencer.*

```
struct _intbytes {          // Data structure to convert a 16-bit
   unsigned char hb;        // integer value to 2 bytes
   unsigned char lb;        // for frequent data manipulation
};
typedef union {
   unsigned int i;
   struct _intbytes b;
}uIntBytes;                 // Declare a new data type
unsigned MterminalCount[3]; // Terminal count for motors
uIntBytes temp;             // Temporary 16-bit variable
unsigned MPCAReload[3];     // PCA reload value corresponding to RPM
void PCASqrWave(void) interrupt 6 using 0  // PCA interrupt on P89C51RD2
{
   if(CCF0) {               // PCA module 0 ISR handler
       CCF0 = 0;            // Clear PCA overflow flag
       M1Clk = ~M1Clk;      // We need to toggle CLK pin ourselves
       temp.b.lb = CCAP0L;  // Read current counter value
       temp.b.hb = CCAP0H;  // and advance it by time
       temp.i += MPCAReload[XMOTOR];       // Required by current speed
       CCAP0L = temp.b.lb;                 // Reload newly computed
       CCAP0H = temp.b.hb;                 // 16-bit value
       MterminalCount[XMOTOR]—;      // Maintain the current position
       if(MterminalCount[XMOTOR]) == 0){   // Terminal count reached
           MotorEvents[XMOTOR].EvStop = 1; // Trigger STOP event
           StopMotor(XMOTOR);              // Stop X-motor
       }
   }
}
void InitPCA(void)
{
// Initial PCA counter module 0 for square wave generation
   CCAPM0 = 0x48;      // Software timer with interrupt for square wave
   CL = CH = 0;        // Clear the free-running counter to begin with
   Temp.i = MPCAReload[XMOTOR] = DEF_X_SPEED;
   CCAP0L = temp.b.lb;
   CCAP0H = temp.b.hb;
   CMOD = 0x01;                        // Osc/6, enable interrupt
   CR=1;                               // Run the counter
   EC = 1;                             // Enable PCA interrupt
   EA=1;                               // Enable the global interrupt flag
}
void RunMotor(char motor)
{
   switch(motor) {
       case XMOTOR:                    // Enable module 0 interrupt flag
       {
           EnableMotor(XMOTOR);        // Switch on winding current
           CCAPM0 = 0x49;              // Enable software timer interrupt
       }
   }
}
void StopMotor(char motor)
{
   switch(motor) {
       case XMOTOR:                    // Disable module 0 interrupt flag

       {
           DisableMotor(XMOTOR);       // Switch off winding current
           CCAPM0 = 0x48;              // Disable interrupt and stop clock
       }

   }
}
```

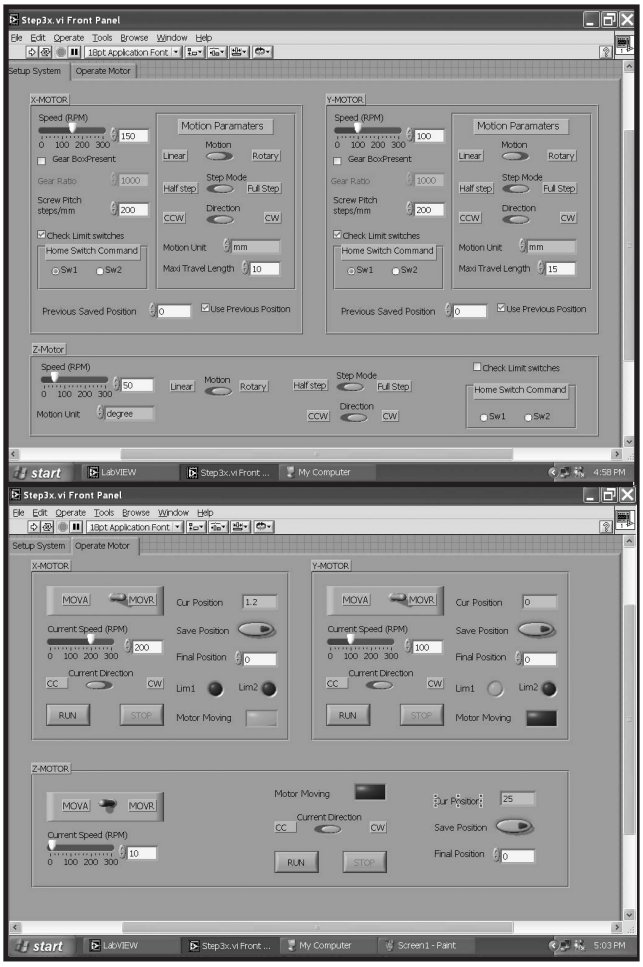


Photo 1—*We created the application so lab technicians could remotely control their equipment. Coding in LabVIEW enables us to deploy the application on Windows and Linux/Unix platforms.*

as a tabbed dialog box. One tab is used for setting the default driver's parameters. The other tab controls the driver. All the parameters declared in the motor data structure can be either configured or changed via the controls on the control panel. This enables you to control the optics equipment via the GUI.

The LabVIEW software generates remote commands in accordance with the selected options for a motor. For instance, if you alter the motor's speed, the software will generate a `SPED:X:100` command and store it in a buffer.

It would also generate command strings that correspond to the rotational direction. If you click Run command button, the software will generate a MOV command knowing the final position requested and mode of movement. It would periodically send a `POSI:X?` query to obtain the current position while moving.

EASY INTEGRATION

Try implementing a stepper motor driver the way we did. Using a motor state machine should help you design state machines for your future applications.

If you have the money, consider IAR System's VisualSTATE, which will automatically generate code based on the state machine. This UML-based tool enables you to develop state machines in the user-friendly manner. It could also generate ANSI C code for the state machine that you could easily link to other firmware modules. VisualSTATE provides you with a compact and efficient API developed specifically for the MCS-51 microcontroller.

You can easily integrate the VisualSTATE with or without a real-time operating system. The tool can also generate documentation that you'd find useful during the code development and maintenance processes. An interactive VisualSTATE simulator allows you to execute a state machine model transition by transition and see how it reacts to events. Project management, prototyping, and product integration are a cinch.

Prajakta Deshpande holds a B.S. from Sagar University, India and an M.C.A. from Barkatullah Vishwa Vidyalaya, Bhopal, India. She is a lecturer in computer science. Prajakta can be reached at ppd@nettaxi.com.

Praveen Deshpande holds a bachelor's degree in electronics engineering from the Regional Engineering College (REC), Nagpur, India. He is currently working as a senior scientific officer in the field of distributed data acquisition and control. His interests include real-time operating systems and distributed control. You may contact him at ppd@cat.ernet.in.

PROJECT FILES

To download the code, go to ftp.circuitcellar.com/pub/Circuit_Cellar/2005/178.

RESOURCES

IAR Systems, "IAR visualSTATE Concept Guide," ver. 4, 1999.

Intel Corp., "MCS 51 Microcontroller Family User's Manual," 272383-002, 1994.

Keil, CX51 compiler information, www.keil.com/support/man/docs/c51/.

SCPI Information, SCPI Consortium, www.scpiconsortium.org.

Philips Semiconductors, "Comprehensive Product Catalog," 9397 750 11146, vol. 5, April 2003.

SOURCES

AT89C2051 Microcontroller
Atmel Corp.
www.atmel.com

VisualSTATE
IAR System
www.iar.com

CX51 Compiler
Keil Software
www.keil.com

LabVIEW
National Instruments
www.ni.com

P89C51RD2 Microcontroller
Philips Semiconductors
www.semiconductors.philips.com

L297 Stepper motor controller
STMicroelectronics
www.st.com

Network GPIB Controller

The GPIB protocol enables you to connect various pieces of test equipment. This team built a low-cost eZ80F91-based network GPIB controller for controlling equipment over Ethernet.

Test equipment such as oscilloscopes, power supplies, and signal generators is often used in manufacturing and compliance applications that require you to control and automate its behavior. Back in the 1960s, Hewlett-Packard recognized this requirement and invented a generic bus protocol that allows you to connect and control test equipment. Today the protocol is called the general-purpose interface bus (GPIB), although it's also known as IEEE 488.1.

GPIB is a multipoint, 8-bit parallel bus that uses a three-wire handshake to acknowledge each data byte. The bus is organized in a master-slave arrangement in which at least one controller is responsible for directing bus communication and a number of talkers that send data and listeners that receive data. Every device on the bus can serve any combination of these three roles. The bus allows for a total of 15 devices with up to 2 m of separation between them and a maximum bus length of 20 m. The maximum nominal transfer rate of the bus is 1 MBps, although there are nonstandard enhancements that can allow it to operate faster.

GPIB, which is supported on a large percentage of test equipment, has proven to be an extremely popular protocol. But the rise of the Internet has made network-enabled electronics all the rage. This reality has also affected the test equipment industry.

A lot of test equipment now includes built-in Ethernet interfaces with embedded web servers and other network control interfaces in addition to (or completely replacing) the GPIB interface. The main advantages of network-enabled control are that the cabling costs are much lower and there aren't inherent distance limitations.

In addition, embedding a web server or client application allows you to control the equipment with any PC that has a

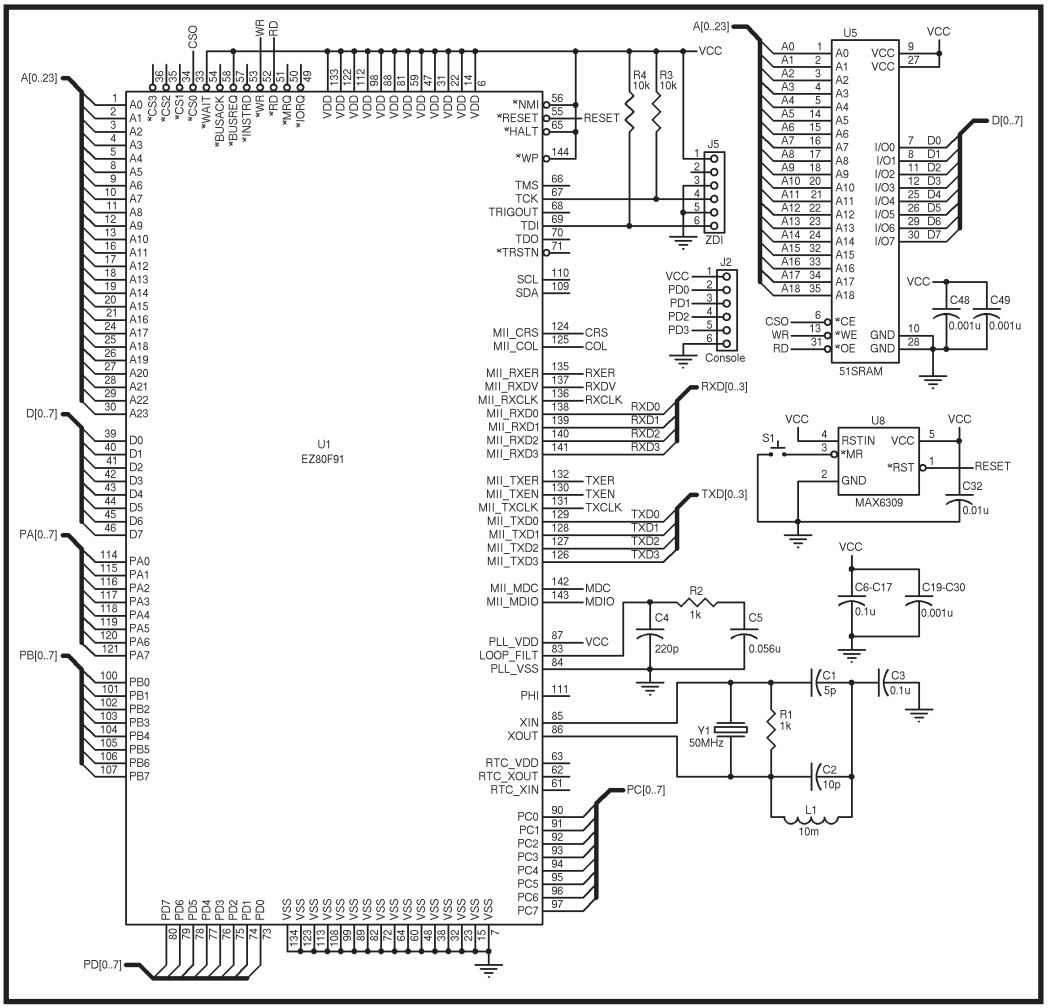


Figure 1—*The brains behind the network GPIB controller is an eZ80F91 microcontroller running at 50 MHz. A MAX6309 programmable reset device monitors the supply voltage and provides a stable reset. Run-time memory for the software is provided by 512 KB of SRAM.*

web browser. You don't have to install the client software first.

A few devices enable you to control equipment with only a GPIB interface over an Ethernet connection. These devices suffer from two major drawbacks. First, they're unnecessarily expensive, which can put them out of reach for many designers and small companies. Second, they don't embed the client application for controlling the GPIB equipment, so you must install software on each PC. We designed a network GPIB controller that solves both of these problems. Our low-cost solution makes it easy to embed client applications.

CONTROLLER HARDWARE

We used an eZ80F91 microcontroller to integrate a number of critical functions into a single device. The GPIB driver is the heart of the controller (see Figure 1). It implements an IEEE 488.1-compliant GPIB bus controller in the software by way of the eZ80F91 microcontroller's external GPIO lines. The eZ80F91 microcontroller also incorporates an embedded Ethernet MAC, which helped reduce both the cost and complexity.

The server software in the controller was built on top of ZTP, the free TCP/IP protocol stack provided by Zilog. We selected the XML remote procedure call (XML-RPC) protocol to be the main means of control between the remote client applications and the controller. XML-RPC is a simple protocol based on XML. It's transported via HTTP. The XML-RPC's main advantages are that it's lightweight and well supported under a number of programming environments.

To demonstrate the ability to embed a client application in the controller and use it to control GPIB equipment, we also implemented a GUI application in Java for the Tektronix TDS-210 oscilloscope. The application has the ability to graphically display waveforms as well as set and control operation, all via the oscilloscope's GPIB connection. The Java application uses the XML-RPC protocol to control the operation of the GPIB bus via the controller.

The primary hardware design criterion for the network GPIB controller was that it must be inexpensive. To accomplish this goal, we focused on a design that integrated as many functions as possible in a minimum number of devices.

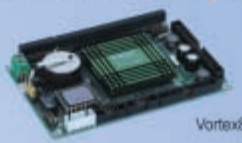
The eZ80F91 microcontroller is the heart of the system because of its high level of integration. It includes the primary flash memory storage for the software, an integrated 10/100 Ethernet MAC, address and data buses for external run-time memory, and ample GPIO pins.

Although dedicated GPIB controllers such as the industry-standard Texas Instruments TMS9914A and NEC Electronics uPD7210 are easy to use, they're expensive and require additional real estate on the board. The eZ80F91 microcontroller is the perfect solution because it can implement the GPIB controller function in software. The GPIO pins on the eZ80F91 microcontroller make it easy to interface to the GPIB bus signals. Cutting out the dedicated GPIB con-

x86 Embedded Processor Module & Single-Board-Computer

Mity-SOC-1
386 Embedded system module
1 RS-232, 1 RS-232/485, 16 GPIO
IDE, FDD, RTC, Parallel, Watchdog
2MB RAM, 2.56" X 1.77" (Optional 4MB)
$65.00 ea. Single unit
$50.75 ea. Qty. 100 pricing

Mity-Mite
386 Embedded system module
1 RS-232, 1-RS-232/485, 16 GPIO,
Ethernet, IDE, RTC, Watchdog, K/B
4MB RAM, 3.14" X 1.96"
$100.00 ea. Single unit
$78.00 ea. Qty. 100 pricing


Vortex86-6082
Vortex86 Embedded system module
1 RS-232, 1-RS-232/485, Parallel, USB
Ethernet, IDE, VGA, K/B, Mouse, RTC,
Watchdog, 128MB RAM, 3.94" X 2.60"
$262.00 ea. Single unit
$182.00 ea. Qty. 100 pricing

Vortex86-6071
PC/104 Vortex86 Embedded SBC
3 RS-232, 1-RS-232/485, Parallel, USB, IDE
Audio, Ethernet, CRT/LCD, RTC, Watchdog
K/B, Mouse, 128MB RAM, 3.77" X 3.54"
$308.00 ea. Single unit
$214.00 ea. Qty. 100 pricing

ICOP-6027VE
3.5" 386 Embedded SBC
CRT/LCD, 1 RS-232, 1 RS-232/485, K/B
Parallel, DiskOnChip, IDE, FDD RTC,
Watchdog, 4MB RAM, 4.01" X 5.67"
$222.00 ea. Single unit
$154.00 ea. Qty. 100 pricing

ICOP-6050
PC/104 386 Embedded SBC
1 RS-232, 1-RS-232/485, 1 parallel, K/B
DiskOnChip, IDE, FDD, RTC, Watchdog
4MB RAM, 3.77" X 3.54"
$142.00 ea. Single unit
$99.00 ea. Qty. 100 pricing

Vortex86-6075
Vortex86 Embedded SBC
1 RS-232, Parallel, 3 USB Ethernet, IDE,
VGA, K/B, Mouse, RTC, Video-in, TV-out
Audio, Watchdog,128MB RAM, 4.37" X 5.24"
$266.00 ea. Single unit
$185.00 ea. Qty. 100 pricing

Vortex86-6047
3.5" Vortex86 Embedded SBC
3 RS-232, 1-RS-232/485, Parallel, USB, IDE
Audio, Ethernet, CRT/LCD, RTC, Watchdog
K/B, Mouse, 128MB RAM, 4.01" X 5.67"
$326.00 ea. Single unit
$226.00 ea. Qty. 100 pricing

Supported OS & Development Environment

DOS
Using C/C++, DOS application can be develop to run on all of our processor modules. DSock, a TCP/IP library for DOS, is provided to develop application with Internet connectivity. Sample implementation for BOOTP/DHCP, FTP, SMTP, HTTP, TELNET & TALK are available for download from our Web site. www.dmp.com.tw/dsock

Embedded Linux
Linux application can run on all of our processor modules. We provide X-Linux, an embedded Linux kernel based on the current popular distribution. X-Linux is a head-less kernel approx. 3MB in size. It includes Linux Kernel 2.4.18, SysLinux Loader, BusyBox Shell, FTP4ALL, udhcp Client, WN HTTP, glibc & Web based administration.

Windows CE .NET
Vortex86 BSP for Windows CE .NET has been certified by Microsoft Windows CE .NET BSP certification program. Windows CE .NET applications can be develop using Embedded Visual C++, Visual Basic .NET & Visual Studio .NET with Compact .NET Framework library. Please check our Web site for Windows CE .NET SDK information.

iCOP
intelligent control on processor
ICOP Technology Inc.
Tel: (626) 444-6666 Email: info@icoptech.com
URL: www.icoptech.com

Microsoft® Windows® Embedded Partner
iCOP is a Gold-Level-Partner of Microsoft Windows Embedded Partner program. Gold WEP has been created by Microsoft to identify companies with demonstrated expertise to support Windows Embedded Technologies.

Contact us for custom design & OEM/ODM services.
Product and service names mentioned herein are the trademarks of their respective owners. Not responsible for error. Prices are subject to change without prior notice.

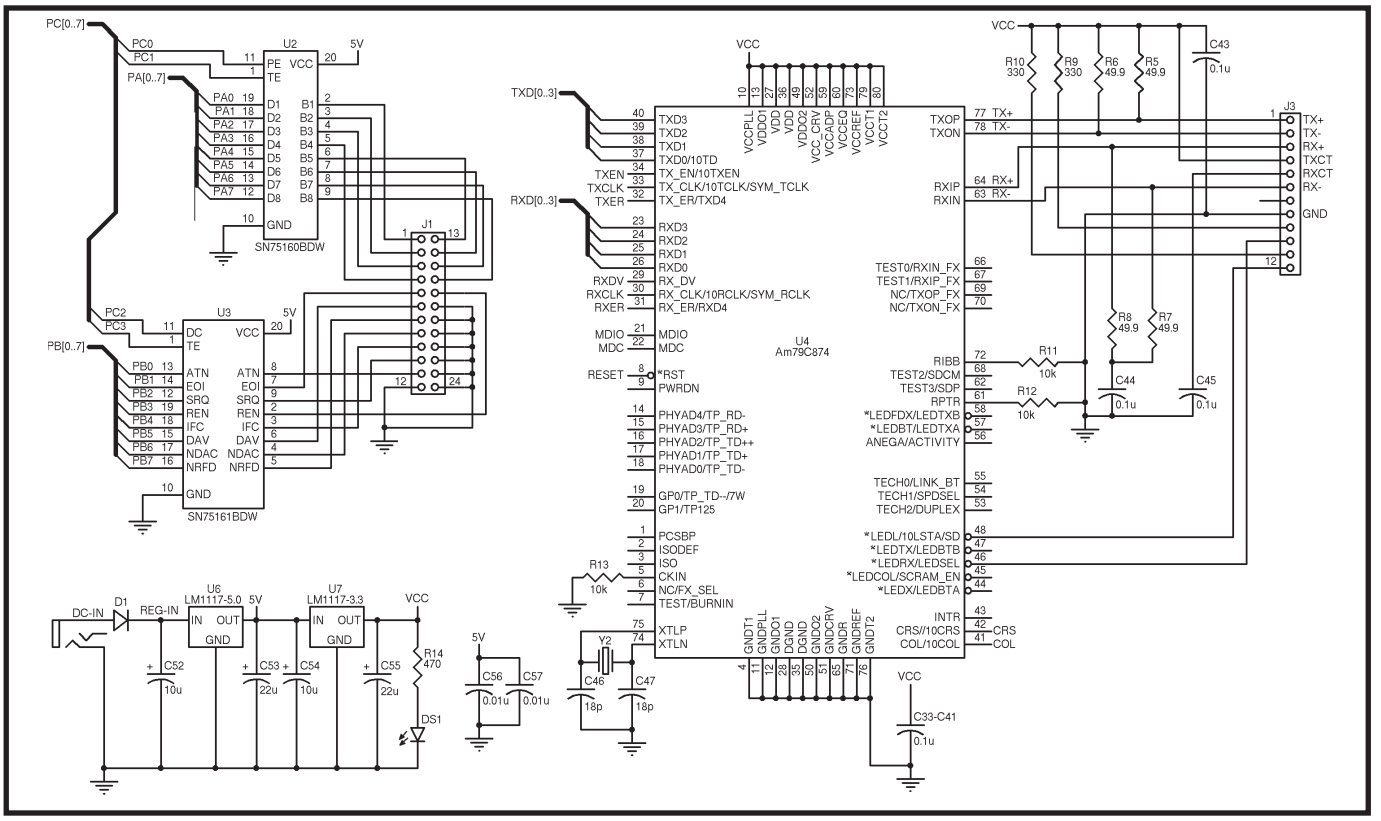


Figure 2—*The Am79C874 is an Ethernet 10/100 PHY device that connects directly to the eZ80F91's media-independent interface (MII) on one side and an RJ-45 jack (with integrated magnetics and LEDs) on the other. The SN75160 and SN75161 provide some state decoding logic as well as line drivers that are electrically compliant with GPIB. The 5- and 3.3-V power supplies are standard fare.*

Embedded Ethernet
only $98 qty 1

- 10Base-T Ethernet
- 186 Processor @ 40 MHz
- DOS w/ Flash File System
- Hardware Clock / Calendar
- 512K DRAM & 512K Flash
- Console / Debug Serial Port
- 16 Digital I/O lines
- Optional DiskOnChip
- 5V DC Power
- Compact 3.75" x 2.50"
- (2) Serial Ports
- (2) 16-bit Timers
- Watchdog Timer

NEW! picoFlash

Expand with the pico-I/O

- 32 Digital Inputs
- 20 Digital Outputs
- 11 Channels of 12 Bit A/D
- C/C++ & Quick Basic Drivers

Call 530-297-6073 Email sales@jkmicro.com
On the web at www.jkmicro.com
JK microsystems

NEW ZigBee™ Modules
XBee™ & XBee Pro™
(Actual Size)
MaxStream XBee www.maxstream.net
Interchangeable
MaxStream XBee PRO www.maxstream.net
Lowest Cost
Highest Performance
-Fast To Market
-Easy To Use
-Free Support
-FCC/CE Ready
Longer-range wireless products also available
Call and order your development kit today!
MaxStream
Toll Free: (866) 765-9885
www.maxstream.net

troller not only reduces cost, but it gives you a measure of investment protection because any updates to the GPIB protocol can be made to the system in the field via a software update.

The GPIB bus signals require line drivers external to the eZ80F91 microcontroller. We chose two commonly used Texas Instruments parts for this task: SN75160 and SN75161 transceivers. The former, which drives the GPIB data bus, is interfaced to the eZ80F91 microcontroller via GPIO port A. The latter, which drives the GPIB control and handshaking signals, is interfaced to the eZ80F91 microcontroller via GPIO port B. Both devices have control signals that are interfaced via four pins on GPIO port C.

The integrated 10/100 Ethernet MAC on the eZ80F91 microcontroller reduces the cost and parts count. An MII bus connection interfaces the MAC to an external PHY, in this case the Advanced Micro Devices Am79C874 PHY, which we chose because it's commonly available and Zilog already produces a driver for the ZTP protocol stack. An RJ-45 jack with integrated magnetics and LEDs rounds out the Ethernet interface (see Figure 2).

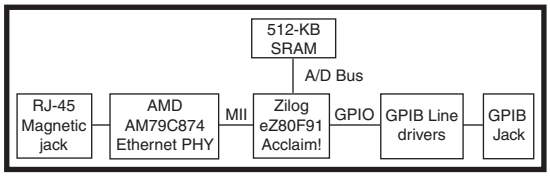


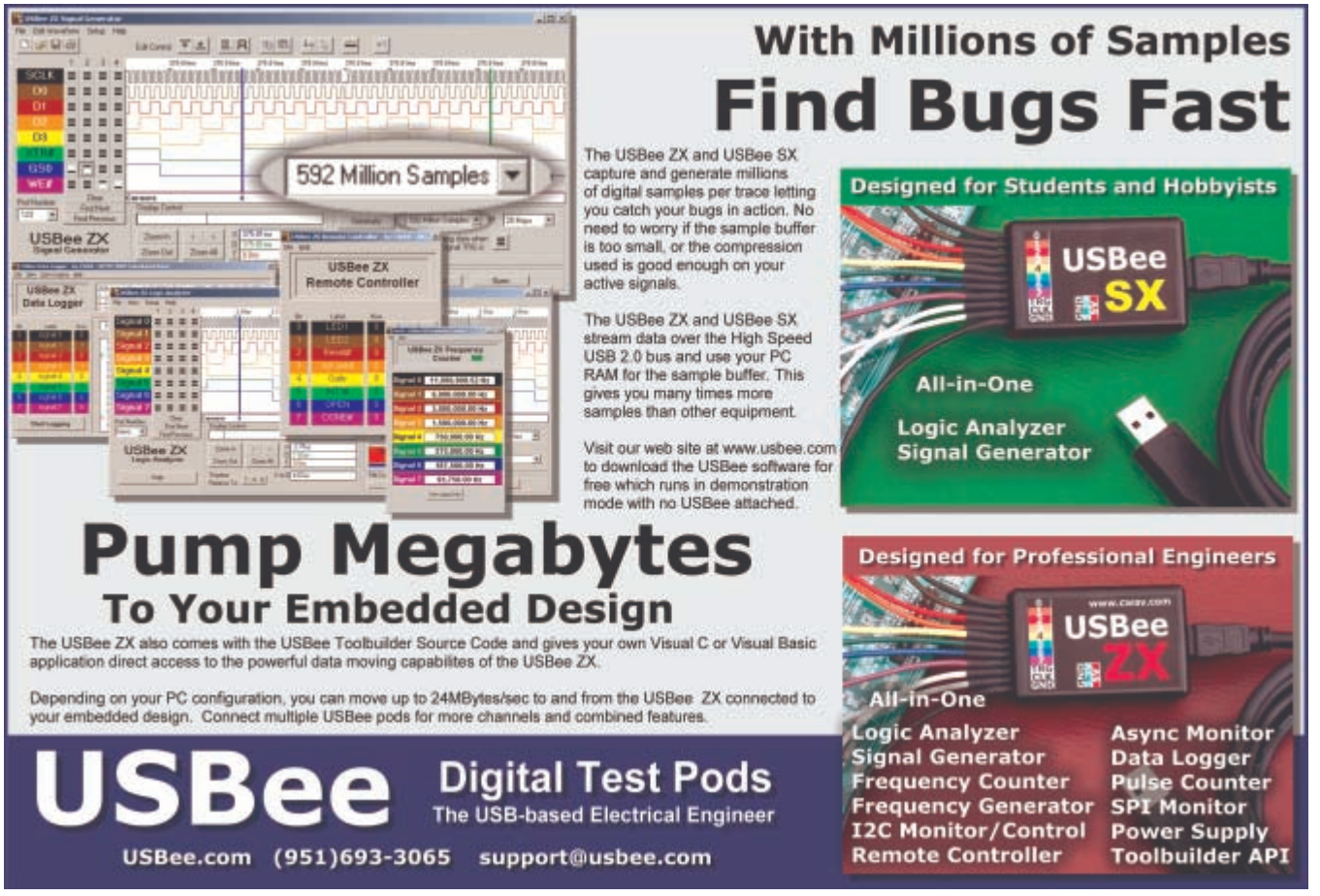
Figure 3—*We satisfied our hardware design goal of having a highly integrated, low-cost system. The eZ80F91 microcontroller performs all of the important functions in the system. The other devices serve only to interface it to the outside world.*

The eZ80F91 microcontroller includes 256 KB of integrated flash memory that provides you with more than sufficient room to store the software image. The software boots out of this flash memory and copies itself to external SRAM for execution. This is done to provide maximum run-time performance and to give the network stack and server plenty of heap and stack space in which to run (see Figure 3).

To accelerate the development schedule and to minimize initial costs, we prototyped the design with the eZ80F91 development kit because it contains the same processor and 10/100 Ethernet PHY as this design. We constructed an external circuit on a small prototype board for the GPIB line drivers. Because the development kit exposes the eZ80F91 GPIO pins on a set of 0.1" headers, it was easy to interface the circuit to the development kit via a standard ribbon cable. To connect the GPIB bus to the line drivers, we sacrificed a GPIB cable by cutting the connector off on one end to expose the wires inside.

SOFTWARE DESIGN

The GPIB bus consists of eight signals for data and eight for handshaking. The maximum data transfer rate is specified to be 1 MBps, but because each data byte on the bus is acknowledged, the speed of the bus transactions actually depends on the speed of


With Millions of Samples
Find Bugs Fast
592 Million Samples
USBee ZX
USBee ZX Remote Controller
The USBee ZX and USBee SX capture and generate millions of digital samples per trace letting you catch your bugs in action. No need to worry if the sample buffer is too small, or the compression used is good enough on your active signals.
The USBee ZX and USBee SX stream data over the High Speed USB 2.0 bus and use your PC RAM for the sample buffer. This gives you many times more samples than other equipment.
Visit our web site at www.usbee.com to download the USBee software for free which runs in demonstration mode with no USBee attached.
Designed for Students and Hobbyists
USBee SX
All-in-One
Logic Analyzer
Signal Generator
Pump Megabytes
To Your Embedded Design
The USBee ZX also comes with the USBee Toolbuilder Source Code and gives your own Visual C or Visual Basic application direct access to the powerful data moving capabilites of the USBee ZX.
Depending on your PC configuration, you can move up to 24MBytes/sec to and from the USBee ZX connected to your embedded design. Connect multiple USBee pods for more channels and combined features.
Designed for Professional Engineers
USBee ZX
All-in-One
Logic Analyzer
Signal Generator
Frequency Counter
Frequency Generator
I2C Monitor/Control
Remote Controller
Async Monitor
Data Logger
Pulse Counter
SPI Monitor
Power Supply
Toolbuilder API
USBee
Digital Test Pods
The USB-based Electrical Engineer
USBee.com (951)693-3065 support@usbee.com

Gain Your Freedom
Lose the Wires

Wireless Solutions For Embedded Control and Connectivity

Rabbit Semiconductor, the leader in embedded networking, has gone wireless. Network freely without the restraints of cables. Add security with our optional SSL and AES software modules.

Bluetooth® Application and Add-On Kit

The Bluetooth Application Kit includes everything needed to develop a Bluetooth application including RCM3100 RabbitCore, Bluetooth Radio Module, prototyping hardware, and development software. In addition, Bluetooth Add-On Kits provide Bluetooth capability for a number of different RabbitCore models.

Bluetooth Add-On Kit $99

Bluetooth Application Kit $339

Wi-Fi® Application and Add-On Kits

The Wi-Fi Application Kit includes everything required to develop a Wi-Fi Application including RCM3100 RabbitCore, LinkSys Wi-Fi card, prototyping hardware, and development software. The Wi-Fi Add-On Kits provide Wi-Fi capability for a number of different RabbitCore Modules. Both the Application Kit and Add-On Kits allow placement of the Wi-Fi up to 12" from the RabbitCore.

Wi-Fi Add-On Kit $199

Wi-Fi Application Kit $599

GPRS / M2M Application Kits

These kits provide all of the tools needed to create a GPRS enabled embedded application including a RabbitCore Module, prototyping board, GSM modem, Integrated Development Environment, and royalty-free TCP/IP stack, with sample programs and libraries to simplify development. The M2M kit provides additional tools to facilitate machine-to-machine communication.

Kit Price $649

Kit Price $699

Free Reference Book With Kit!

Embedded Systems Design using the Rabbit 3000 Microprocessor free with wireless kit purchase.

www.rabbitsemi4wireless.com

$49 Value

RABBIT Semiconductor

2932 Spafford Street, Davis, CA 95616
Tel **530.753.8400**

9167

the devices being addressed. The eZ80 has more than enough speed to keep up with the real-time demands of the GPIB bus.

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0	Talk	Listen	A5	A4	A3	A2	A1

Table 1—*A simple byte command byte format can address a device in a single byte. Address 31 is reserved for the "untalk" and "unlisten" commands.*

Each GPIB bus has one controller in charge (CIC). It's the function of the controller to send commands and specify who may talk on the bus. Because our controller acts in place of a PC and runs the client application, it acts as the CIC for the bus.

The handshaking signals pertinent to our design include: Interface Clear (IFC), Attention (ATN), Service Request (SRQ), Not Ready for Data (NRFD), Data Available (DAV), Not Data Accepted (NDAC), and End or Identify (EOI). The controller takes control of the bus by pulling the IFC signal low for at least 100 µs upon initialization. To keep the bus quiescent during periods of inactivity, we keep ATN asserted, which forces all of the other devices off the bus. A device can assert SRQ if it needs service.

Addresses in GPIB commands are 5 bits along. One bit is unused. The other 2 bits specify whether the address is a talk address or a listen address. The range is from one to 31 (see Table 1). But because primary address (PAD) 31 is reserved, it leaves a total of 31 addresses for use on the bus (zero through 30). By convention, PAD 0 is reserved for the controller, but this isn't mandatory.

There are two types of data on the GPIB bus: commands and data. Asserting ATN when you place the command data on the bus identifies the former. The CIC is the only device allowed to send commands and assert ATN. Commands are individual bytes used to specify which devices should listen and which should talk. Command bytes also exist that "untalk" or "unlisten" all devices on the bus. Any number of command bytes can be sent while ATN is low. All devices on the bus must monitor these commands.

To send data, the driver first needs to set "talk enable" on the transceivers to ensure that you're driving the correct signals onto the bus. ATN is then driven low to force all of the devices into a listen state. After that, you can send commands to "unlisten" and "untalk" the devices so that the bus is in a known state. The controller then addresses itself as a talker by sending its talk address followed by the listener address.

Handshaking transmission data is the same for command and data bytes. First, the driver makes sure that NRFD isn't asserted, indicating that all of the addressees are ready for data. The driver then places the data on the bus and asserts DAV. When all of the devices have accepted the data, NDAC will go high and the driver can de-assert DAV. This cycle is repeated for each byte sent.

There are three different ways to terminate the transmission: assert EOI on the last byte, send a line feed (0x0A), or do both (see Photo 1). In order to receive data, the driver must first address another device as a talker. Again, the driver begins by asserting ATN, and it then "untalks" and "unlistens" all devices on the bus. The driver then addresses itself as the listener and addresses the new talker. Following this, the driver can de-assert ATN and begin to receive data from the talker.

Handshaking received data takes on the opposite role from a send. After ATN is de-asserted, the bus has been turned over to another device as the talker, so the transceivers must be switched out of Talk Enable mode. The driver then raises NRFD and asserts NDAC to indicate that it's ready to receive data. After DAV goes low, the driver asserts NRFD and retrieves the data from the bus. This cycle repeats for each byte of data. Again, the transmission is terminated by an EOI or a line feed.

The GPIB driver provides a standard board-level interface API to the XML-RPC server. These routines mirror those of the well-known API used by GPIB libraries for various platforms like Windows and Linux. The functions map directly into the network control functions provided by the XML-RPC server to the client software. Because of time constraints, we implemented the functions for demonstrating the board's functionality (`SendIFC`, `SendCmds`, `SendDataBytes`, `Send`, `ReceiveSetup`, `RcvRespMsg`, and `Receive`) rather than the entire standard API.

XML-RPC SERVER

The network GPIB controller requires a network server with a protocol that can manage GPIB requests and responses from remote clients. The primary requirements for the protocol are that it must be simple, it must operate reliably over an IP network, and it must be built on top of the ZTP TCP/IP protocol stack. We considered the implementation of a proprietary server and protocol, but because of its simplicity, broad platform support, and ease of implementation, we selected the XML-RPC for the task.

The XML-RPC protocol performs remote procedure calls using HTTP as a transport and XML as the data encoding. Requests are issued via a standard HTTP-POST request, normally to the /RPC2 URI path on the server. The server always sends a response to a request. An example XML-RPC request/response sequence for a GPIB send function is

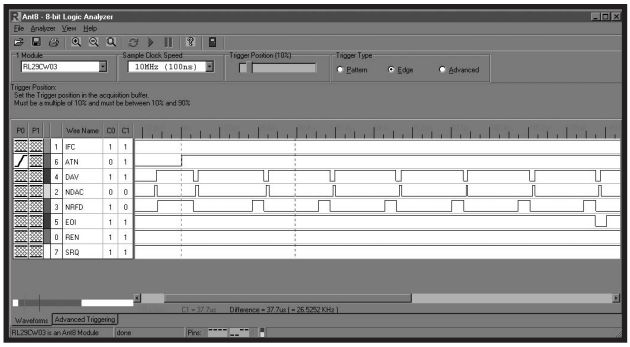


Photo 1—*Take a look at the GPIB control lines. ATN is de-asserted so these are data cycles. EOI signals the end of the transaction.*

shown in Listings 1 and 2.

The availability of a web server in the ZTP package made it extremely easy to integrate our XML-RPC server because all of the HTTP processing was handled automatically. The XML-RPC server hooks into the ZTP web server by overriding the default POST processing function.

We considered using an off-the-shelf XML parser based on either the standardized DOM or SAX programming interfaces. Because parsers based on them are generic, they have fairly high run-time memory and code space requirements for an embedded application. Therefore, we implemented a custom XML parser that focuses on the XML parsing functionality needed by XML-RPC. The XML parser is designed to minimize the memory footprint as much as possible by parsing the XML in place inside the request buffer.

The procedure calls supported by the XML-RPC server map almost exactly to the standardized GPIB API defined by IEEE 488.2. This makes it extremely easy for someone familiar with GPIB to write client applications. The only difference is the mechanics of how procedures are called, not the procedure call format itself.

CLIENT SOFTWARE

We developed a client application to demonstrate and test the capabilities of the networked controller. Our goal was to provide a familiar user interface for a GPIB device connected to our controller. In our case, this was a Tektronix TDS-210 digital oscilloscope. An application with a simple graphical user interface exposing enough functionality to perform most of the basic operations of an oscilloscope would do the trick.

We had three basic requirements for the programming language. The first was a free and functional XML-RPC client library. Preferably, it would be open source. One of the main reasons we chose XML-RPC was because of its prevalence, so this requirement didn't narrow the field too much. A quick 'Net search returned free libraries for all of the popular languages, including but not limited to C/C++, Java, Microsoft.NET, Perl, Python, PHP, and TCL.

The second requirement was free and mature development tools, including a compiler/interpreter and a windowing tool kit. Again, this didn't shorten the list much because it only excluded .NET. Finally, to exploit one of our device's primary value-added features, we needed it to be able to be embedded in the eZ80F91 microcontroller's HTTP server as a web browser applet. Of course, this last requirement made Java the clear choice.

In a nutshell, the client applet presents the oscilloscope's screen and controls translating user interactions into GPIB calls and screen updates. The oscilloscope's GPIB interface is like a console that accepts ASCII commands and queries. Commands and queries are sent from the client to the oscilloscope as RPCs, invoking the GPIB `Send` method. Answers to queries are read back with RPCs invoking the GPIB `Receive` method. For example, to query the vertical scale (i.e., volts per division) of channel one, the `"CH1:SCALE?"` string is sent with a GPIB `Send` RPC. To read the response to the query, a subsequent GPIB `Receive` is sent. Listing 3 shows the steps in Java language.

Photo 2 shows the complete system running the Java client application. The development board is in the foreground with the ZPAK debugger connected. Immediately behind the board is the GPIB driver circuit with a GPIB cable wired to it. The other end of the GPIB cable is connected to the back of the TDS-210 oscilloscope on the right. The monitor in the back shows the Java client application running on a Windows PC.

So far, the network GPIB controller has been tested against only the TDS-210, a National Instruments PCI-GPIB+ adapter, and an older ISA adapter with a TMS9914A GPIB controller on it. However, it performed

Listing 1—*XML-RPC requests are defined by a standardized XML schema that contains a series of nested tags defining what procedure call to execute and its parameters. The example request demonstrates the* `gpib.Send(2,"CH1:SCALE?",1)` *call.*

```
<?xml version="1.0"?>
<methodCall>
  <methodName>gpib.Send</methodName>
  <params>
    <param>
        <value><int>2</int></value>
        <value><string>CH1:SCALE?</string></value>
        <value><int>1</int></value>
    </param>
  </params>
</methodCall>
```

Listing 2—*The XML-RPC example response is also XML. It uses a schema similar to that used by the requests. The response either returns a single return value or a standardized fault response. The example response returns three values:* `ibsta=0`, `iberr=0`, *and* `ibcntl=0`.

```
<?xml version="1.0"?>
<methodResponse>
  <params>
    <param>
      <value>
        <struct>
          <member>
              <name>ibsta</name>
              <value><int>0</int></value>
          </member>
          <member>
              <name>iberr</name>
              <value><int>0</int></value>
          </member>
          <member>
              <name>ibcntl</name>
              <value><int>0</int></value>
          </member>
        </struct>
      </value>
    </param>
  </params>
</methodResponse>
```

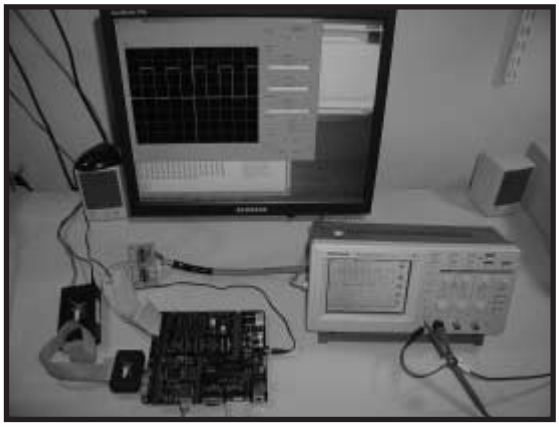


Photo 2—*The functioning prototype was built around the eZ80 Acclaim! development kit. Note that the waveform displayed on the oscilloscope matches the waveform on the client application. All of the eZ80F91 microcontroller's GPIO lines were conveniently available at headers. The ZPAK made downloading and debugging code across the network a snap.*

ification. The hardware work is fairly obvious: turn the prototype into a circuit board and put it into an enclosure.

We achieved a high level of integration by choosing an eZ80F91 microcontroller with a built-in Ethernet MAC and incorporating a firmware-only GPIB controller. Using the freely available ZTP TCP/IP stack and open-source software components enabled us to cut development time and costs. By choosing XML-RPC as the messaging layer, we almost have off-the-shelf support for clients in all of the popular programming languages, any of which can be easily client-enabled with a minimal amount of glue logic.

well and with excellent data throughput performance in all cases. The eZ80F91 microcontroller is well suited to an embedded network control interface such as this one. It will save us board space and additional cost with its integrated flash memory and Ethernet MAC devices as well.

We've passed some significant milestones in the development of the network GPIB controller. Although we now have an extremely functional prototype, we'd still like to complete several tasks. As for the software, we need to finish the GPIB driver because it doesn't implement the full IEEE 488.1 controller spec-

Listing 3—*In this GPIB RPC query sequence, the client sends a query asking the oscilloscope for its channel one vertical scale settings with an XML-RPC* `gpib.Send()` *call. Following this, the output is retrieved with a* `gpib.Receive()` *call and then parsed.*

```
//Create a client object connected to the server
XmlRpcClient xmlrpc = new XmlRpcClient("http://192.168.1.51/RPC2");
// Build the gpib.Send() parameters list
Vector params = new Vector();
params.addElement(new Integer(2));              //GPIB primary address
params.addElement("CH1:SCALE?".getBytes());     //String to send (in
                                                //binary format)
params.addElement(new Integer(1));              //Termination value
//Make the Send remote procedure call
xmlrpc.execute("gpib.Send", params);
//Build the gpib.Receive() parameters list
Vector params = new Vector();
params.addElement(new Integer(2));              //GPIB primary address
params.addElement(new Integer(500));  //Receive a maximum of 500 bytes
params.addElement(new Integer(1));              //Termination value
//Make the Receive remote procedure call and store the response
Hashtable response =
    (Hashtable) xmlrpc.execute("gpib.Receive", params);
//Receive returns the query response in a hash table with the key "buffer."
//Extract the binary-formatted answer and store it in a byte array.
byte[] buffer = (byte[]) response.get("buffer");
//Convert the answer back to an ASCII string. This will be the V/div
//setting of channel one in floating point (e.g., 100E-3 for 100 mV).
String str = new String(buffer, "US-ASCII");
```

Editor's note: Ron, Patrick, and Scott won First Prize in the Zilog 2004 Flash Nets Cash Design Contest for their Network GPIB Controller. Visit www.circuitcellar.com/magazine/contests.htm for more information.

Ron Battles has been involved with networking since installing Novell networks more than 14 years ago while attending the University of Texas at Austin. He has written network drivers and embedded code for Thomas-Conrad, Intel, NetSpeed, and Cisco Systems on a variety of platforms and processors. He is currently a software engineer at Cisco Systems. You may contact him at ronbo@austin.rr.com.

Patrick Jackson received a B.S.E.E. degree from the University of Texas at Austin. He has been developing embedded signal processing and computer networking systems for 10 years. Patrick is currently a software engineer at Cisco Systems. You may contact him at pej@austin.rr.com.

Scott Shumate is a software architect at Cisco Systems, where he's currently designing Metro Ethernet products. He has been involved in the development of embedded communication systems for nine years in the areas of ATM, DSL, and Ethernet. Scott holds a B.S. degree in computer engineering and an M.S. degree in electrical engineering from the University of Kansas. You may contact him at scott@shumatech.com.

PROJECT FILES

To download the code, go to ftp.circuitcellar.com/pub/Circuit_Cellar/2005/178.

RESOURCES

eZ80F91 MCU information, www.zilog.com/products/partdetails.asp?id=eZ80F91.

F. M. Hess, "Linux-GPIB 3.1.98 Documentation,"2003, http://linux-gpib.sourceforge.net/doc/index.html.

HPIB Tutorial, http://ftp.agilent.com/pub/mpusup/pc/iop/hpibtut/ib0_toc.html.

XML-RPC information, www.xml-rpc.com.

SOURCES

Am79C874 Ethernet PHY
Advanced Micro Devices, Inc.
www.amd.com

TDS-210 Digital oscilloscope
Tektronix
www.tektronix.com

SN75160 and SN75161 GPIB Transceivers
Texas Instruments, Inc.
www.ti.com

eZ80F91 Microcontroller
Zilog, Inc.
www.zilog.com

Speech Synthesis with SpeakJet

The SpeakJet sound synthesizer is an affordable alternative to recorded voice synthesis. This month Jeff shows you how to add voice and sound features to your embedded project.

Computer-generated speech can sound extremely natural, especially if you have the money to spend and the computing and storage capabilities of a PC to work with. But sometimes don't you just want to be able to add speech output to an embedded design without having to deal with the high costs and loss of resources?

In most cases you probably don't need your system's voice to be as soothing as HAL's in *2001: A Space Odyssey*, so long as it's easily understandable. In this column I'll show you how to bring a little life to your designs with the affordable Magnevation SpeakJet sound synthesizer.

SPEECH SYNTHESIS

Most speech synthesis models can be broken down into two parts: front-end and back-end components. The front end takes raw text and performs the tokenization task. This is a process whereby the text is analyzed at the syllabic level and a phonetic translation is assigned. It may also involve designating points of inflection based on punctuation. The back end of a speech synthesis system takes the phoneme stream from the text preprocessor and creates the actual sound output.

There are two approaches to creating sound output: concatenative and formant. The concatenative approach strings together prerecorded segments of speech in a number of different ways. Depending on the application, the database of utterances may be composed of complete sentences or broken into individual phonemes (minimum distinctive sounds) depending on the application. An alternative to a database of phonemes is a diphone, which is a recording of the transition between phonemes. Note that the number of possible transition sounds goes up as the square of the number of phonemes. All possibilities, however, aren't needed.

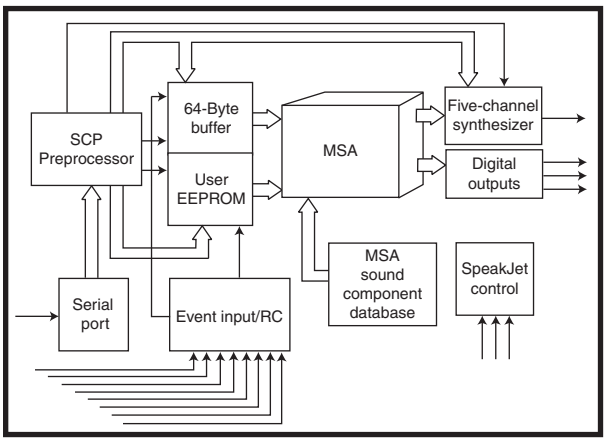


Figure 1—*The mathematical sound architecture (MSA) module is the main component in the SpeakJet. The serial port enables you to enter allophone data for immediate speech or sound output. You can save the data in nonvolatile memory for subsequent output triggered by an event input.*

Formant synthesis doesn't use any prerecorded samples. It uses an acoustic source-filter model in which frequency, voicing, and noise levels vary over time. Multiple formant paths are often combined to more closely imitate the complexity of the human vocal tract. Because a formant database contains instructions for producing speech instead of actual samples, it's much smaller than the concatenative database.

Votrax, General Instruments, and Silicon Systems were some of the first players using analog formant technologies. Various prerecorded content techniques were also manufactured by National Semiconductor (Digitalker), Texas Instruments (LPC), and OKI Semiconductor (ADPCM).

LINGUISTICS

You can create intelligible speech if you keep in mind a few basic linguistic concepts concerning the relationship between text and speech: the same letter in different words can often sound completely different, so think in terms of sounds, not letters, when converting words into speech; even an allophone's sound may be influenced by its neighbor; and finally, to avoid monotonic speech, use textual punctuation to indicate where to alter the default parameters (pitch, rate, bend, and volume).

Phonemes can be broken down into consonants and vowels. The English alphabet has six vowels (including Y) and 20 consonants. Obstructing your vocal tract with your tongue or lips to create a discontinuous noise source, as with the consonants "t" and "b," produces the latter. Consonants also may have a continuous sound source from the vibrations of vocal cords, as with the fricatives *f*, *h*, *s*, *sh*, *th*, *v*, and *z*.

On the other hand, all vowel sounds have a relatively unrestricted path through the resonant cavities of the vocal tract, throat, nose, and mouth.

The continuous sound source from vibrating vocal cords is altered by your tongue's placement and the shape of your lips. Although short and long vowels can have a single vowel sound, some combinations use a transition between two vowel sounds. These diphthongs combine to create a single allophone such as the *a* sound in the word "make."

Although generic rules for pronunciation can cover most situations, you must pay attention to more than just the letters. An exception dictionary will help you fix mispronunciations. But automatic text-to-speech conversion is a complex process that's beyond the scope of this article. For more information on the topic, refer to my column, "Next-Generation Text to Speech," in which I describe the Winbond WTS701, one of the most complete external text-to-speech processors on the market (*Circuit Cellar*, 158, September 2003). Now let's focus on the SpeakJet synthesizer.

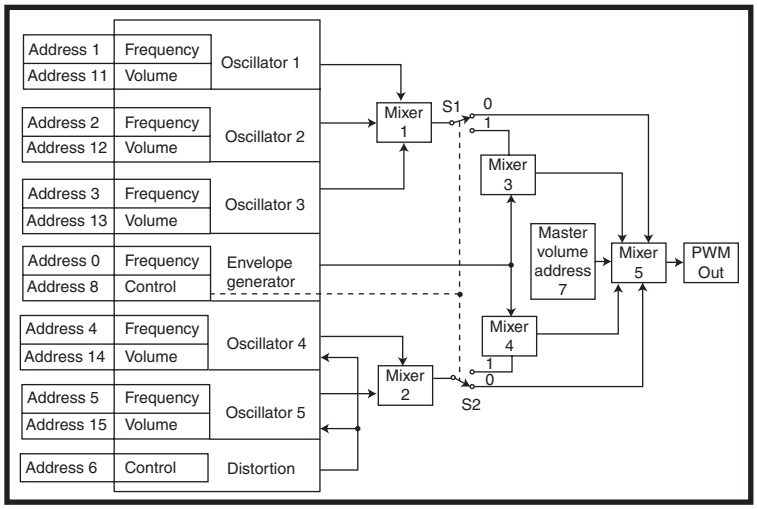


Figure 2—*The synthesizer is composed of a number of oscillators. The oscillator outputs can be mixed in a number of ways to reproduce the required speech or sound output.*

SpeakJet SYNTHESIZER

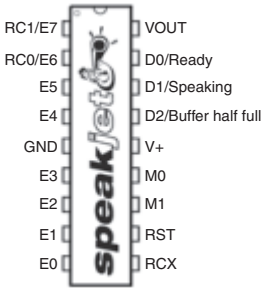
Have you ever heard of Magnevation? No, I haven't switched subjects here. It has nothing to do with bullet trains or superconductors. Magnevation began designing peripherals in support of the OOPic and other microcontrollers. A robot builder at heart, Magnevation's owner Jeff Richeson developed the SpeakJet for roboticists who long to give their creations a voice. And thanks to the technological advances in microcontrollers, an inexpensive alternative to recorded voice synthesis is now possible.

The SpeakJet speech and sound synthesizer uses a mathematical sound architecture (MSA) to model the human vocal tract. Seventy-two discrete allophones are used to reproduce speech. The mathematical instructions used by the MSA to reproduce the sound of each allophone are stored in the database.

The SpeakJet isn't a text-to-speech converter. You must provide it with a series of allophones, or it must be stored on-board in nonvolatile memory. Instructions for the allophones are pulled from the database. This allows the MSA to calculate the sound components of each allophone and any transitional sounds linking the allophones together (see Figure 1).

In addition to providing an unlimited vocabulary, the MSA can be tweaked to give speech more emotion than Ben Stein. Sprinkling control commands anywhere along the allophone string will modify the rate, pitch, bend, and volume of each allophone. (One of the potential phases of a text-to-speech converter is to look at punctuation and add modifiers to add emphasis where required.)

The SpeakJet reproduces allophones using an array of oscillators, so you can produce practically any sound (see Figure 2). In fact, in addition to the allophones stored in the database, a number of useful sounds are also preprogrammed, such as DTMF (telco) tones and an array of beeps, boops, and chirps.



Pin number	Description	Functional details
1	RC1/E7	Event input 7
2	RC0/E6	Event input 6
3	E5	Event input 5
4	E4	Event input 4
5	GND	Ground
6	E3	Event input 3
7	E2	Event input 2
8	E1	Event input 1
9	E0	Event input 0
10	RCX	Serial input TTL (0 V to V_{CC})
11	RST	Master reset
12	M1	Mode select 1 (data rate configure)
13	M0	Mode select 0 (Demo mode)
14	V+	Power input 2 to 5.5 VDC
15	D2/Buffer half full	Data out 2 (external)/buffer half full (internal)
16	D1/speaking	Data out 1 (external)/speaking (internal)
17	D0/Ready	Data out 0 (external)/ready (internal)
18	VOUT	Voice output

Figure 3—*The 18-pin SpeakJet can produce canned output based on eight event inputs in Stand-alone mode. The RCX serial input permits dynamic control via a microcontroller.*

SERIAL CONTROL PROTOCOL

Although you can use the SpeakJet as a stand-alone system, you must be able to program canned requirements into its nonvolatile user memory. The pinout in Figure 3 includes an RCX pin. It's a TTL serial input using an 8N1 format from 2.4 to 19.2 kbps. All serial SCP characters are ASCII-printable, enabling communication via a terminal program such as HyperTerminal.

Command	Function	Description
dH	Store memory type	0 = direct register 32 = nonvolatile control registers
dJ	Store memory address	0–15 (for direct registers) 00–255 (for control registers)
dN	Store memory	Decimal data to store
R	Clear buffer	Clears both buffers
S	Stop enunciating	Halts extracting additional buffer data
T	Start enunciating	Continues extracting data from buffer
V	Acknowledge	Say power-up phrase
W	Hard reset	Resets SpeakJet
X	Exit	Exits Command mode

Table 1—*You can use these commands in Command mode to communicate directly with the SpeakJet's control registers.*

The data directed to the MSA's 64-byte input buffer is a 3-byte string composed of an escape character (/) followed by two hexadecimal characters. This directs the MSA to select one of the 256 preprogrammed sound segments.

About half of the MSA's 256 preprogrammed sound segments (127–254) are reserved for allophone and sound codes. Of the remaining codes, 95 are reserved for the future, while the first 32 (0–31) have significant meaning. You can use these codes to execute pauses, add emphasis to sounds, or control output bits.

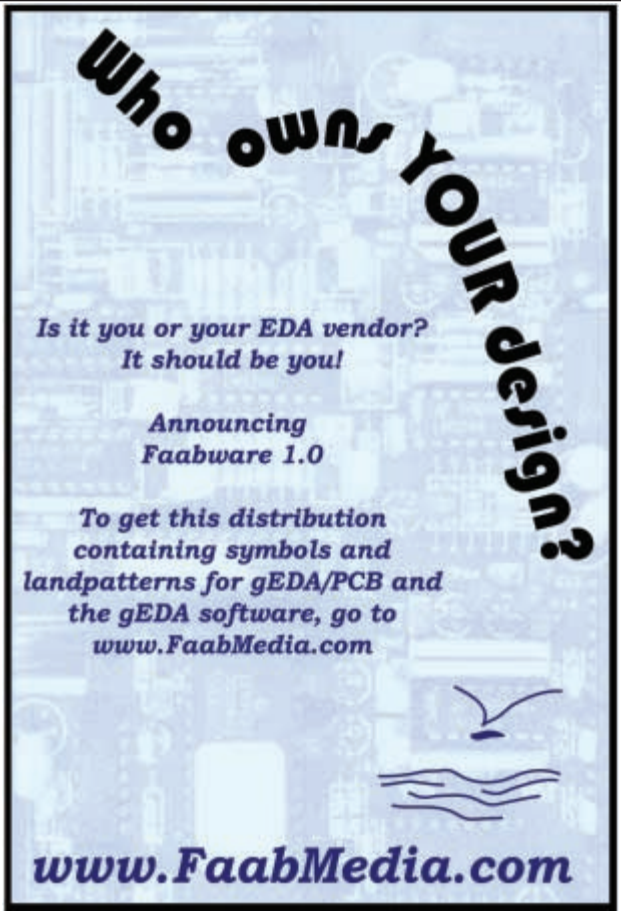
The serial control protocol (SCP) initiates direct control of the SpeakJet's internals. Using the escape character along with a single decimal digit (0–7) will request Command mode. Because a SpeakJet module can be assigned an address (0–7) if it matches the received command, it will enter Command mode and redirect any data following it into the 16-byte command buffer.

The recognized commands in Table 1 are used to store data in a control register, start and stop execution of the data in the 64-byte input buffer, and exit Command mode. Data can be stored in direct registers (synthesizer registers 0–15) and nonvolatile registers (0–255).

The SpeakJet has 256 nonvolatile control registers composed of phrase pointers, phrase storage, and chip control areas. Up to 16 user-programmed phrases can be stored on the SpeakJet. Phrases are stored consecutively in registers 16 through 236, which form the total character storage for all phrases and may limit the number of phrases you can store. Registers 0 through 15 hold pointers that indicate the beginning of each phrase (up to 16). This means each phrase can be as small or as large as necessary (as long as no phrase extends beyond register 236).

The top 19 registers control power-up reset phrase, mode, node, and event registers. As you can see in Table 2, the bits in register 237 indicate which canned phrases (0–15) should be played at reset. The bits in register 238 define the function of several digital input and output bits as well as Auto Silence, which turns down the volume after reaching an empty buffer. Register 239 defines the SpeakJet's node address (0–7) and can define the polarity of the three output bits. The last 16 registers define the use of the eight event inputs (two reg-

NEW!
Fishing for signals?
Unique USB-powered scope adapters.
Best value-for-money in USB PC Scopes!
Keep one with your laptop for field servicing.
Swordfish™ 1-ch 10-bit 40MS/s scope, data logger, FFT spectrum analyzer, voltmeter and frequency meter. Unique lightweight USB hand-held-probe turns your laptop into a 5MHz scope.
PS40M10
Only $275!
Stingray™ 2-ch 12-bit 1MS/s scope, data logger, FFT spectrum analyzer, voltmeter, frequency meter and sig-gen/trigger. Unique lightweight USB dual-channel adapter turns your laptop into a 500kHz scope.
DS1M12
Only $220!
Free EasyScope/EasyLogger software. DLL's for easy 3rd party applications. LabView examples provided. WinCE and Linux drivers on request.
Saelig Company Inc.
1-888-7SAELIG
www.saelig.com/ad/fishcc505.htm
info@saelig.com
UNIQUE PRODUCTS FROM AROUND THE WORLD: www.saelig.com/osscour.htm


Who owns YOUR design?
Is it you or your EDA vendor?
It should be you!
Announcing
Faabware 1.0
To get this distribution containing symbols and landpatterns for gEDA/PCB and the gEDA software, go to www.FaabMedia.com
www.FaabMedia.com

isters for each event input). The even-numbered registers act on rising transitions. The odd-numbered registers act on the falling transitions of each event input. These registers define which phrase will be spoken. They also determine whether or not the buffer is cleared before speaking the phrase (blows away anything it is presently saying).

MSA

The MSA operates the five-channel synthesizer according to parameters stored in the database. Each value received via the serial port (except escape commands redirecting data to the SCP buffer) causes the MSA to execute one of its database functions. The MSA command set consists of a set of actions (0–31), allophones (128–199), and sound effects (200–254).

Note that only about half of the 256 possible commands are implemented. The rest are reserved (32–127). Also note that the reserved values include the characters used in the SCP. Most of the first 32 MSA action commands modify the way in which the MSA sets up the five-channel synthesizer. Many of the parameters reference a default value. Some action commands can alter these defaults. This has the effect of changing the volume, speed, pitch, and bend parameters used with the five-channel synthesizer on an allophone-by-allophone (or sound-by-sound) basis.

Additional action commands can act on the output port, repeat a sound a certain number of times, and call any of the 16 canned phrases. Keep in mind that the phrases may contain any of the MSA commands; and so, extremely powerful nested routines may exist (up to three deep).

5-CHANNEL SYNTHESIZER

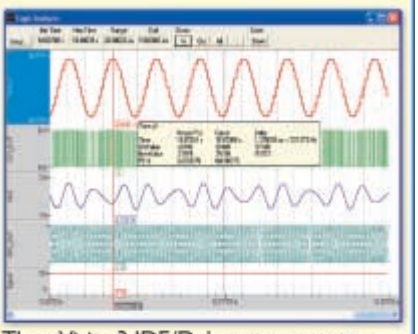
It's hard to imagine that all of the SpeakJet's output is simply modulation on a 32-kHz PWM carrier. A two-

Register 237: Power-up/reset phrase	
Bit 7	0 = Play 1 = Silence
Bit 6	Always 1
Bit 5	Always 1
Bit 4	Always 1
Bit 3	Bit 3 of phrase pointer number to call on power-up/reset
Bit 2	Bit 2 of phrase pointer number to call on power-up/reset
Bit 1	Bit 1 of phrase pointer number to call on power-up/reset
Bit 0	Bit 0 of phrase pointer number to call on power-up/reset
Address 238: PortCTL, Auto Silence, RC/TTL Mode	
Bit 7	Reserved
Bit 6	Reserved
Bit 5	RC0/E6 (0 = RC input 1 = TTL)
Bit 4	RC1/E7 (0 = RC input 1 = TTL)
Bit 3	Auto Silence (0 = off disable 1 = enable)
Bit 2	D2/Buffer half full (0 = phrase control 1 = output)
Bit 1	D1/Speaking (0 = phrase control 1 = output)
Bit 0	D0/Ready (0 = phrase control 1 = output)
Register 239: PortCTL, Auto Silence, RC/TTL Mode	
Bit 7	Always 1
Bit 6	Bit 2 of SCP node address
Bit 5	Bit 1 of SCP node address
Bit 4	Bit 0 of SCP node address
Bit 3	Always 1
Bit 2	D2/Buffer half full (0 = off 1 = on)
Bit 1	D1/Speaking (0 = off 1 = on)
Bit 0	D0/Ready (0 = off 1 = on)
Register 240/2/4/6/8 and 250/2/4: Event inputs E0–E7 rising transition	
Bit 7	Say phrase (1 = yes 0 = no)
Bit 6	Do not clear input buffer before phrase (1 = yes 0 = no)
Bit 5	Execute a call phrase (1 = yes 0 = no)
Bit 4	Do not clear control buffer before call phrase (1 = yes 0 = no)
Bit 3	Bit 3 of phrase pointer to say
Bit 2	Bit 2 of phrase pointer to say
Bit 1	Bit 1 of phrase pointer to say
Bit 0	Bit 0 of phrase pointer to say
Register 241/3/5/7/9 and 251/3/5: Event inputs E0–E7 falling transition	
Bit 7	Say phrase (1 = yes 0 = no)
Bit 6	Do not clear input buffer before phrase (1 = yes 0 = no)
Bit 5	Execute a call phrase (1 = yes 0 = no)
Bit 4	Do not clear control buffer before call phrase (1 = yes 0 = no)
Bit 3	Bit 3 of phrase pointer to say
Bit 2	Bit 2 of phrase pointer to say
Bit 1	Bit 1 of phrase pointer to say
Bit 0	Bit 0 of phrase pointer to say

Table 2—*The chip control registers create a powerful Stand-alone mode. Events (rising, falling, or both) can generate prestored phrases or control output bits.*

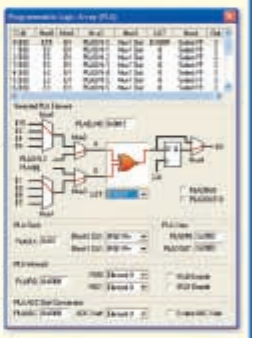
KEIL SOFTWARE

Keil Professional Embedded Development Solutions provide the tools you need for your 8051, 251, C16x, and ARM projects. Tools from Keil help you efficiently create, test, & complete your embedded projects.



The µVision3 IDE/Debugger supports virtual target simulation and includes a software logic analyzer that graphically displays variables and I/O registers.

The µVision3 Development Environment provides complete device simulation and includes peripheral dialogs that aid development of software drivers.



µVISION 3

Supported Architectures

ARM7

Keil development tools for ARM support a wide variety of devices from the most popular chip vendors.

C166

Keil development tools for the XC16x, C16x, and ST10 support over 50 devices from Infineon and STMicroelectronics.

Cx51

Keil development tools for the 8x51 support over 600 different derivatives including devices with extended address space and enhanced instruction sets.

Contact us today for a Free Demo CD

800-348-8051

www.keil.com

pole, low-pass filter is sufficient to remove this carrier and let the modulation through. However, if a passive filter is used to remove the 32 kHz, an amplifier will be necessary to get the signal back up to a useable level.

Each oscillator is capable of running at approximately 4 kHz (using an 8-kHz sampling rate). The oscillator banks are broken into three groups. Oscillators 1, 2, and 3 are used as voiced sound sources (zz). Oscillators 4 and 5 can be distorted by white noise and are useful as voiceless sound sources (ff). The groups can be mixed individually as well as collectively. In addition, an envelope generator can apply a saw, sine, triangle, or square wave to modulate the volume of each group independently.

PHRASE-A-LATOR

The SpeakJet requires a string of allophones to speak text directly. Magnevation offers an application written for the PC to help prepare your allophone strings. The Phrase-A-Lator is a powerful sound designer that enables you to build and test phrases, sounds, and control sequences. The SpeakJet can interface serially with your PC if you add an RS-232-to-TTL translation driver (MAX232) or use the SpeakJet activity center PCB (see Photo 1).

The Phrase-A-Lator software also allows you to program the internal EEPROM with your newly built phrases and sound effect and control codes. You can customize sounds through the internal five-channel synthesizer.

The user interface offers six miniature applications: Phrase Editor, Synthesizer Control, EEPROM Editor, Dictionary Editor, Event Control, and Data Rate/Detuning. The Phrase-A-Lator uses a text-based dictionary to recall the phonetic equivalent of common words. The text file is easily edited with any text editor.

You can add new words designed with the Phrase Editor to the dictionary with a single click (see Photo 2). Use the EEPROM Editor application to create the 16 canned user phrases and program them to the SpeakJet's EEPROM. You can also insert sound effects and control code into user phrases intermixed with allophones. This is where the first 32 MSA commands can create powerful operations. You can make your own sound effects with the Synthesizer Control application, which gives you access to the internal five-channel sound synthesizer. The Event Control application helps you determine how the eight inputs will use the canned user phrases.

SpeakJet PROJECTS

Take full advantage of the SpeakJet's

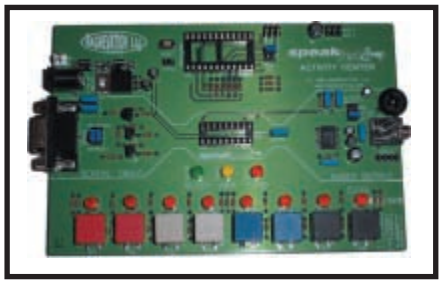
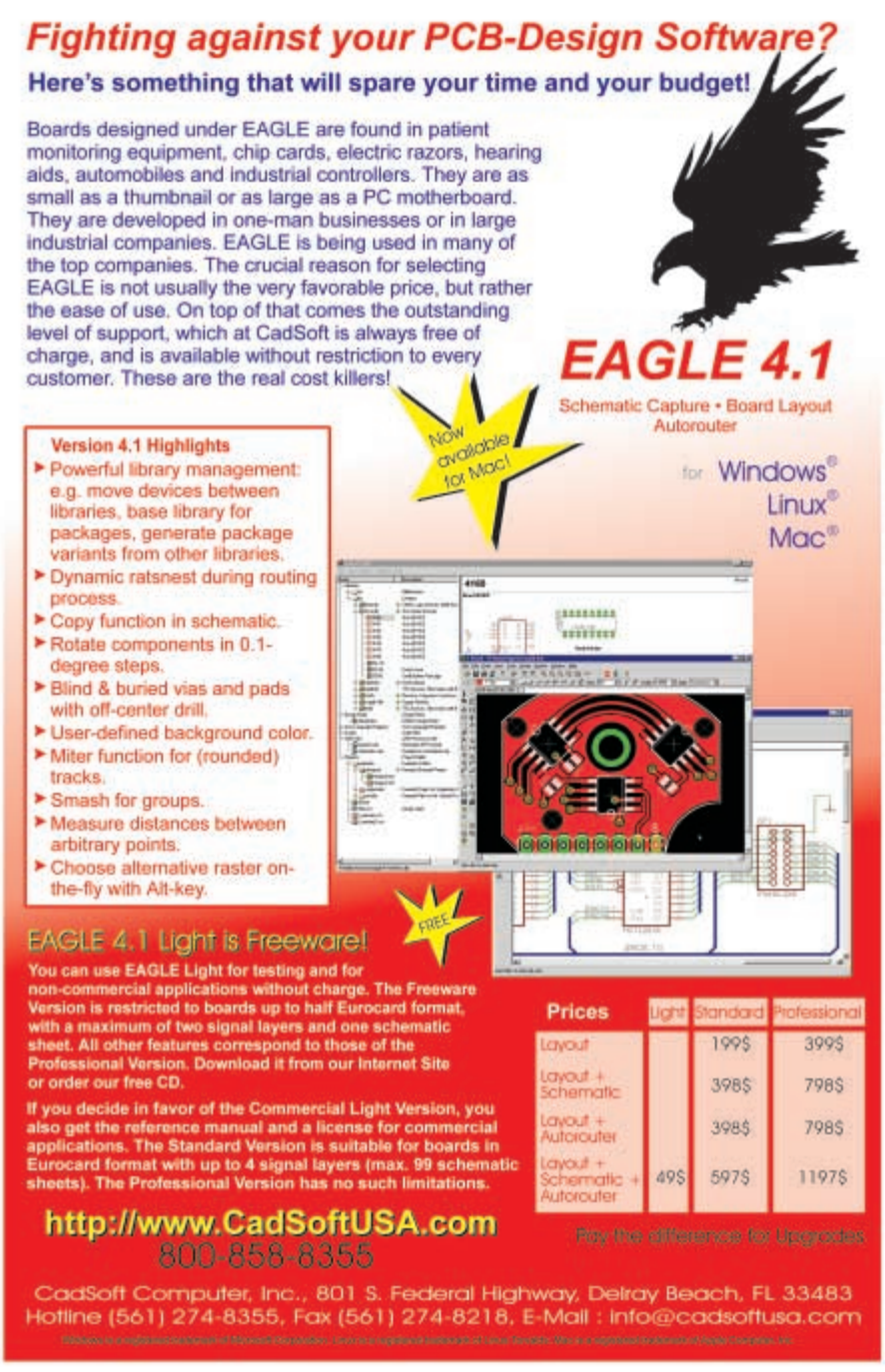


Photo 1—*Even though you can use Magnevation's activity board as a stand-alone device, it has a serial port so you can connect it to a PC. The free Phrase-A-Lator software complements the activity board's hardware for a total development environment.*


Fighting against your PCB-Design Software?
Here's something that will spare your time and your budget!
Boards designed under EAGLE are found in patient monitoring equipment, chip cards, electric razors, hearing aids, automobiles and industrial controllers. They are as small as a thumbnail or as large as a PC motherboard. They are developed in one-man businesses or in large industrial companies. EAGLE is being used in many of the top companies. The crucial reason for selecting EAGLE is not usually the very favorable price, but rather the ease of use. On top of that comes the outstanding level of support, which at CadSoft is always free of charge, and is available without restriction to every customer. These are the real cost killers!
EAGLE 4.1
Schematic Capture • Board Layout
Autorouter
for Windows® Linux® Mac®
Now available for Mac!
Version 4.1 Highlights
Powerful library management: e.g. move devices between libraries, base library for packages, generate package variants from other libraries.
Dynamic ratsnest during routing process.
Copy function in schematic.
Rotate components in 0.1-degree steps.
Blind & buried vias and pads with off-center drill.
User-defined background color.
Miter function for (rounded) tracks.
Smash for groups.
Measure distances between arbitrary points.
Choose alternative raster on-the-fly with Alt-key.
FREE
EAGLE 4.1 Light is Freeware!
You can use EAGLE Light for testing and for non-commercial applications without charge. The Freeware Version is restricted to boards up to half Eurocard format, with a maximum of two signal layers and one schematic sheet. All other features correspond to those of the Professional Version. Download it from our Internet Site or order our free CD.
If you decide in favor of the Commercial Light Version, you also get the reference manual and a license for commercial applications. The Standard Version is suitable for boards in Eurocard format with up to 4 signal layers (max. 99 schematic sheets). The Professional Version has no such limitations.
Prices
Light
Standard
Professional
Layout
199$
399$
Layout + Schematic
398$
798$
Layout + Autorouter
398$
798$
Layout + Schematic + Autorouter
49$
597$
1197$
Pay the difference for Upgrades
http://www.CadSoftUSA.com
800-858-8355
CadSoft Computer, Inc., 801 S. Federal Highway, Delray Beach, FL 33483
Hotline (561) 274-8355, Fax (561) 274-8218, E-Mail : info@cadsoftusa.com

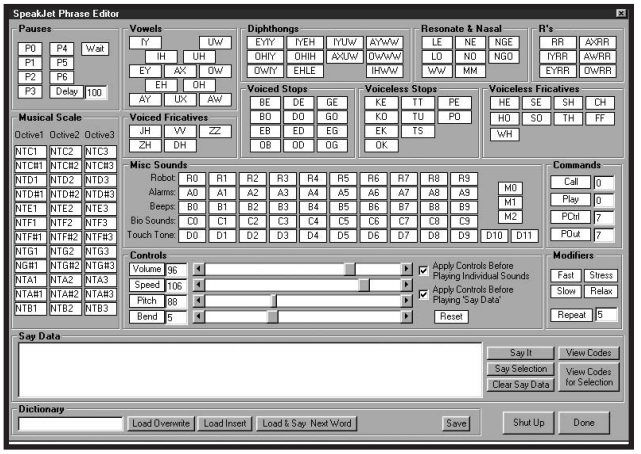


Photo 2—*You can call allophones and sound effects with a simple click of your mouse. You can experiment to determine which allophones should be used for a particular word, phrase, or sentence. You can also tweak the program to make it sound more natural.*

canned user phrase EEPROM area. After it's programmed, the SpeakJet can interact with the real world all on its own. The inputs are triggers that can initiate one (or more) of the canned areas. Remember that in addition to the programmed speech and sounds, the embedded codes can also operate any of the three output lines. Combined with delays, you can create sophisticated output cycling.

You might want to create a security system that monitors an electrical closed loop and responds when the loop is opened. A response might be an audio message or an output that takes your phone system off the hook. External phone line circuitry can route the audio into the phone to dial your phone (using its DTMF capability) and repeat a message, say, 10 times before hanging up.

At this point, neither logic nor decision-making has been implemented, so branching based on various input states isn't permitted. Any project that requires a higher level of decision-making would take additional circuitry and might have to include a microcontroller. Although you can't overlook the power of adding a microcontroller, as a stand-alone sound unit, the SpeakJet is extremely powerful for its size and price.

Use the SpeakJet to bring a little life to your speechless robots. Even something as simple as your system saying "Ready" can instill a level of comfort and confidence. For a doctor, there is nothing like having a patient be able to tell you where it hurts. As a programmer, it's easy to assume you know which code is being executed when in fact the program flow may be in an entirely unexpected area. A good programmer will use a microcontroller's unused outputs as process indicators. You can tell what's happening with an oscilloscope or LED.

Using speech or sound as a debugging tool can be useful but totally offensive. Discretion is a good idea. You might want to implement a way to enable and disable a "chatty" mode .

The SpeakJet is perfect for adding speech and sound capabilities to a product without having to use an MCU. (You may download a sample WAV file I made using the Speakjet from the *Circuit Cellar* ftp site.) However, by taking advantage of its serial communication path, your microcontroller can harness its power.

Jeff Bachiochi (pronounced BAH-key-AH-key) has been writing for Circuit Cellar *since 1988. His background includes product design and manufacturing. He may be reached at jeff.bachiochi@circuitcellar.com.*

PROJECT FILES

To download the speakjet.wav file, go to ftp.circuitcellar.com/pub/Circuit_Cellar/2005/178.

RESOURCE

S. Lemmetty, "Phonetics and Theory of Speech Production," Helsinki University of Technology, 1999, www.acoustics.hut.fi/~slemmett/dippa/index.html.

SOURCE

SpeakJet sound synthesizer and Phrase-A-Lator software
Magnevation
www.speakjet.com

PIC-SERVO
Motion Control
Low-Cost Motion Controllers for Brush, Brushless and Stepper Motors
New PIC-SERVO SC Single Chip Servo Controller!
▪ Controller Chips
▪ Controller Boards
▪ Integrated Amplifiers
Visit our web site for a complete selection of controllers and accessories
www.jrkerr.com
JEFFREY KERR, LLC

USB/ETHERNET DAQ
LabJack UE9
Available now for only ...
$399 qty 1
USB/Ethernet Data Acquisition & Control
* USB 2.0/1.1 and Ethernet
* 14 analog inputs (12- to 16-bit)
* Stream input data up to 50 kHz
* Use with C, VB, LabVIEW, etc.
* Includes DAQFactory Express
* Operates from -40 to +85 deg C
* 2 analog outputs (12-bit)
* 23 digital I/O
* Up to 2 counters (32-bit)
* Up to 6 timers
* Approx. 3" x 7" x 1"
LabJack Corporation, Colorado, USA
info@labjack.com, (303) 942-0228
www.labjack.com

ANDRE LAMOTHE'S
XGAMESTATION
DO-IT-YOURSELF VIDEO GAME SYSTEM
Inspired by the Atari 2600, Apple II & Commodore 64!
SX52 @ 80 MIPS
INCLUDES:
•Assembled XGS Micro Edition Unit!
•Complete Development Kit!
•Tools, Demos & Utilities!
•eBook on Designing the XGS Console!
•Cables and Power Supply Included!
WWW.XGAMESTATION.COM
(925) 736-2098 SUPPORT@NURVE.NET

Connect with USBLab

No parallel printer port? Stuart's USBLab plugs into a USB port and provides you with eight bidirectional data bits, three address bits, a read/write direction control line, and a data strobe. It also has a serial input and output. Problem solved.

Back in the good old days of computing, around 1997, you could connect a new widget to a PC's parallel printer port and control it. All you had to do was write to the parallel port's I/O addresses to make things happen. But now most new computers come with either Windows 2000 or XP operating systems, which don't allow user-level programs to access the printer port. You can find a driver that will allow such access, but it will be expensive, complicated to work with, and won't always work if the parallel port is configured in EPP mode. What can you do with a new computer that doesn't have a parallel printer port?

I designed the USBLab to solve these problems. The USBLab plugs into a USB port and provides eight bidirectional data bits, three address bits, a read/write direction control line, and a data strobe. Plus, it has a serial input and output. The former allows you to program the data rate with binary divisor values so you can obtain nonstandard data rates to match your project, which may not use a multiple-of-standard-data-rates crystal. The serial I/O is TTL, not RS-232, so you don't have to add RS-232 conversion to your microcontroller project.

USBLab CIRCUITRY

The USBLab uses an Atmel ATmega8515 microprocessor (U2) connected to a DLP-USB245 (U1) adapter, which is a circuit board mounted to a 24-pin header that will fit in a standard 24-pin DIP socket (see Figure 1). The DLP-USB245, which has a USB connector, connects to a USB port on your PC. The module looks like a standard serial communication device to the PC, but it stores the incoming data and presents it to an external microcontroller (the ATmega8515) via an 8-bit data bus. The DLP-USB245 has read and write input signals so the external microcontroller can send and receive data.

The DLP-USB245 handles the USB protocol, so you don't have to write any of the messy USB protocol software. It also provides low signals on pin 13 (*RXF), when there is incoming USB data to read, and on pin 14 (*TXE), when the transmit FIFO is ready to accept data. The MCU writes to the FIFO just as it would write to a memory device or register by placing data on the eight data lines (pins 17 through 24) and strobing the *WR signal (pin 15) low. Similarly, data is read by strobing *RD (pin 16) low.

The DLP-USB245 uses an FT245BM IC, which is one of a series of the Future Technology Devices International parts that provides modular functional blocks such as serial I/O and parallel FIFO-based interfaces. The DLP-USB245 combines the FT245BM with a crystal, a USB connector, and EEPROM (to store USB enumeration values).

The ATmega8515 microcontroller reads incoming USB data from the DLP-USB245. The interface uses a simple command opcode followed (in some cases) by data. The opcode tells the ATmega8515 whether to write the data to the parallel port, serial port, or data rate registers. In some cases, the opcode isn't followed by data. For example, the command to read the

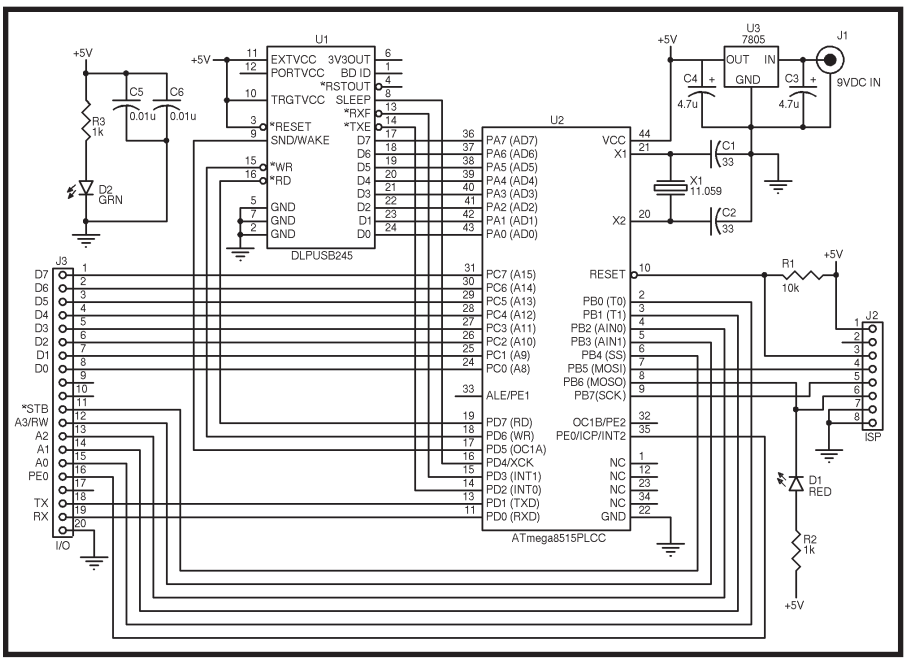


Figure 1—*An ATmega8515 microcontroller controls the USBLab, which requires only three integrated circuits. Connector J2 enables you to in-circuit program the ATmega8515 microcontroller. You may replace J2 with a standard 10-pin connector.*

parallel port doesn't send data; it expects data to be returned.

The ATmega8515 operates from an 11.059-MHz crystal, which provides standard data rates. However, by programming the data rate divisor, you can get nonstandard rates. If you have a specific project for which you want a specific data rate, build the project with a crystal that's a multiple of that rate. The ATmega8515 will operate with a crystal up to 12 MHz.

```
>>>
Board ID: USBLab v1.0
Walking one return code: 4E time: 0.0320000648499
Write addr 0-7 return code: 4E time: 0.0309998989105
Read addr 0-7 return code: 20 23 21 23 22 23 23 23 24 23 25 23 26 23 27 23 4E time:
0.109000205994
PE0 return code: 4E time: 0.140000104904
Serial return data: 30 41 30 42 30 43 30 44 30 45 30 46 30 47 30 48
Set baud Return Data: 4E
Invalid opcode return value: 7F
>>>
```

Figure 2—*The Idle window should show the results of running the TestUSBLab.py program. Timing may vary from these values.*

The circuit is powered by an external supply. A USB device may be powered by the USB bus, but current is limited, and in most cases, the USBLab will be used in conjunction with some kind of self-powered project. To avoid USB power management issues, I made the USBLab self-powered as well.

Power is applied to the USBLab via a coaxial jack (J1) and regulated by a 7805 regulator (U3). A 9-VDC, 500-mA wall wart transformer supplies power. LED D2 indicates when power is applied.

Input/output connector J3 connects to your external project. You can read and write up to eight external registers by decoding the three address lines A0 through A2. Signal PE0 is an extra output signal. Separate commands allow you to set and clear PE0 without affecting the state of D0 through D7 and A0 through A2.

Connector J2 is an eight-pin header that allows the ATmega8515 microcontroller to be programmed in-circuit. This is an in-line header that matches my programmer. You could also use the standard Atmel 10-pin ISP header.

CIRCUIT CONSTRUCTION

I used the PLCC version of the ATmega8515 so the part could be placed in a standard through-hole PLCC socket. The DLP-USB245 was placed in a standard 24-pin, machined-pin IC socket. I don't recommend a leaf-type socket because the DLP-245BM has round pins. You can, of course, wire to the DLP-245BM without using a socket at all.

The expansion connector J3 is a 20-pin (10 × 2) header—the type of header into which you'd plug a ribbon cable. Jack J1 is wired with the center terminal as the positive connection. You can, of course, change that to match a power supply you already have.

C5 is located near the ATmega8515's V_{CC} pin (pin 44). C6 is located near the DLP-USB245 V_{CC} pin (pin 11).

LED D1 should blink when power is applied and the ATmega8515 is programmed. If it doesn't check your connections, make sure that the 5-V supply voltage is correct and that the crystal is connected correctly.

I wrote the TestUSBLab.py checkout program in Python. It uses a communication module called commport.py that I wrote for a previous project. You must have Python and the Python win32 extensions installed on your computer to use TestUSBLab.py.

After installing Python and the Win32 extensions, turn on power to the board and plug it into the USB port. Windows should detect the device. If you're using Windows XP SP1, the DLP communication port driver will probably load automatically. If it doesn't, you can get the driver from the DLP Design's web site. Use the VCP driver.

Connect the expansion connector's pins 18 and 19 (Tx and Rx) with a jumper. Before running TestUSBLab, you must set the COM port to match your system. You can find the COM port by opening the control panel and selecting System, Hardware, and then Device Manager. You should see a COM port in the Ports window that isn't part of your system. If you aren't sure which one it is, check the device manager before and after you plug in the USB cable.

After you identify the COM port number, right click the TestUSBLab.py file, select Edit with Idle, and find the line that opens COM port 7:

```
ch = c.OpenCommPort(7)
```

Change the 7 to whatever COM port the USBLab enumerates as in your system; it will typically be port 3 or 4. It was 7 on my system because I have a com6 device. (Refer to the USB Enumeration sidebar (p. 78) for more information.) Next, save the file and then select the Edit/Run or Run/Run module (depending on your version of Idle) to run the script. You can run the program by double clicking it, but the results would be displayed in a DOS box that disappears after the program is finished. The Idle window should display the results shown in Figure 2.

The data line starting with "Read addr" may vary, but it should be 20 xx 21 xx, 22 xx... The time to complete the tests will vary as well; however, if you get these results back, you'll know the basic hardware is working.

To verify that the expansion port is wired correctly, copy the walking one's code from TestUSBLab.py into a new Python program (keep the imports and port set-up code), and modify it to delay a couple of seconds (with `time.sleep(2)`) after each write. This will allow you to use a logic probe or DVM to verify that the bits are walking through the pins. Or, if you have one, you can use a logic analyzer like I did.

FIRMWARE

The firmware in the ATmega8515 microcontroller loops continuously and monitors the serial port for incoming data and the DLP-USB245 for incoming USB data. The ATmega8515 microcontroller's data bus is configured to access external memory. Because the DLP-USB245 has no address inputs, the ATmega8515 microcontroller's upper eight address outputs are configured to function as general-purpose I/O lines.

If incoming data is found on the serial port, the firmware will write the SendSerial opcode to the DLP-245, then write the received serial byte. If the DLP-USB245 indicates that USB data has been received, the firmware will read the data and process it. The firmware always expects an opcode to be first. If the opcode is SendSerial, WriteParallel, or SetBaudRate, the firmware expects additional data

Opcode value	Meaning
0x00	NOP
0x10	WriteParallel (bits 0–2 are address to write)
0x20	ReadParallel (bits 0–2 are address to read)
0x30	: SendSerial
0x40	SetBaud
0x44	SetPE0
0x48	ClrPE0
0x4C	TransmitNow
0x4D	GetBoardID
0x4E	ReturnAck

Table 1—*The USBLab interprets the opcodes to perform specified actions. Some opcodes are followed by one or more data bytes.*

bytes. One byte is expected for SendSerial and WriteParallel. Two bytes are expected for SetBaudRate.

The firmware checks the opcodes for validity (see Table 1). If an invalid opcode is encountered, a value of 7F is returned to the host. The USBLab won't hang in that case, but if the invalid opcode is caused by a synchronization problem when sending data to the USBLab, you may get some strange results.

The WriteParallel opcode can range from 0x10 to 0x17. ReadParallel opcode can range from 0x20 to 0x27. The lower 3 bits specify which address on the expansion connector are to be written and read. The SetBaud opcode is followed by the high byte of the data rate and then the low byte of the data rate. These values are written directly to the ATmega8515's data rate registers.

The `GetBoardID` command will cause the USBLab to return the text "USBLab" and the board's revision. The string (all ASCII characters) is terminated with a zero byte. Generally, you shouldn't issue the board ID command while the USBLab is receiving serial data because serial data may be inserted ahead of the board ID string. The board ID is intended to identify the USBLab if you have multiple COM devices connected.

Note that the board ID string won't be interrupted with serial or parallel data. Therefore, if your code looks for the text "USBLab"in the returned USB data and then reads until it finds a zero, you can issue and handle the board ID command at any time. The `ReturnAck` command causes the opcode to be returned to the host. The firmware doesn't buffer USB or serial data. If the DLP-USB245 stops accepting data, the firmware will hang and wait for the transmit-ready signal to activate. The firmware object code is contained in the USBLab.hex file. It's an Intel hex format file.

SYSTEM AT WORK

The commport.py code provides simple communication functions that can be used with the USBLab. It's important that the timeout is set like it is in the TestUSBLab.py code because this allows the code to read from the USBLab without having to wait too long.

When processing returned data from the USBLab, you need to process 1 byte at a time because the opcode/data pairs may be broken up into two USB transmissions. In general, you should read an opcode and wait for another byte if no more data is available. You can't guarantee that a single read from the USB COM port will return both bytes of a pair.

I/O timing using the USBLab is a bit different from timing using a parallel port. With the parallel port, your code can manipulate the data bits in line and know when the actual bit changes occur. With the USBLab, the USB interface introduces unknown latency in between writing data and the actual changes on the expansion connector outputs. In addition, the DLP-USB245 has a latency timer for determining how often to transmit stored data to the PC.

You can synchronize the USBLab with your code with the `ReturnAck` command. The USBLab processes commands in the order they're received. By executing a series of parallel write commands followed by a `ReturnAck` command, you can be certain that all the parallel data is written when the `ReturnAck` command is received.

You can do something similar with the send serial command by placing a `ReturnAck` command at the end of the last serial transmit command pair. In this case, the `ReturnAck` command will indicate that the last serial byte was written to the output UART, not necessarily that it was transmitted.

Matching the timing of the serial data with the timing of control inputs is often a problem when you're using a serial port to debug a

USB Enumeration

After a Windows-based PC detects that a USB device has been inserted, it queries the device to see what it is. The query process involves passing several descriptors back and forth, but the result is that Windows obtains a vendor ID (VID) and product ID (PID). These values are used to select the driver that will communicate with the device. Enumeration is the process of obtaining the VID/PID and other information. If Windows can't find a driver, it will ask you for one. The default VID and PID for the DLP-USB245 are 0403 and 6001.

A USB device can use its own driver, or it can enumerate itself as a standard, Windows-recognized device class. These classes include USB mice, keyboards, and mass storage devices. Devices using predefined USB classes are expected to operate according to the protocols for that class, so no driver is needed (or, more precisely, Windows has built-in drivers for them). This is how all the different USB flash memory-based drives can operate without driver software. It's also why those flash memory drives need a driver for Windows 98, which didn't have a predefined mass storage class.

With the right firmware, one device can make use of more than one driver by issuing different VID/PID pairs during enumeration. You could make a device appear as a COM port. Then, by flipping a switch, you could re-enumerate it as a memory-storage device. In fact, so-called "soft"USB devices do exactly that. A soft USB device doesn't contain any permanent storage. The operating code, which is contained in RAM, is lost when you unplug the device or turn off the power. A soft USB device uses a driver that loads the operating code before the device reenumerates as whatever type of device it's supposed to be.

Writing your own USB driver can be a problem because you have to obtain a VID/PID from the USB organization. If you don't, you risk a conflict with a commercial product. USB is intended for products that are produced to the tune of thousands, not one-time experimenter projects. The simplest solution is to use a product such as the DLP-USB245, which comes with drivers, and adapt it to your needs.

project. The USBLab writes received serial data to the DLP-USB245 as it's received, so you can correlate the received serial data with the commands you're sending to the parallel port. To correlate received serial data with the WriteParallel and Set/ClearPE0 opcodes, use the `ReturnAck` command to indicate completion of those commands.

When connecting devices to the expansion connector, make sure you drive data on the data lines only when the USBLab is reading. Use the read/write signal and the data strobe to ensure this. Drive the data bus only when read/write is high and the data strobe is low.

If you fall out of synchronization with the USBLab when you're debugging, your code or the USBLab (or both) will probably confuse the data and opcodes. If that happens, you should reset the USBLab or send a string of NOP or `ReturnAck` commands.

If you don't want to use the USBLab as a COM port, DLP Design has a DLL-based driver that provides essentially the same features. You can call the DLL from any software that's capable of calling a DLL, including Python and, of course, C language. If you're using the DLL driver, you can change the EEPROM values, including the vendor VID and PID, and the latency timer value.

EASY USB

The USBLab provides an easy way to access functionality similar to the old parallel port, but on newer computers and operating systems. In addition, the inclusion of the serial port provides new debug potential for your computer-connected designs. Finally, you can apply the principles described in this article to adapt the DLP-USB245 to your own application. Now you can add an easy USB connection to your next design.

Stuart Ball is an engineer at Seagate Technologies with more than 20 years of experience in embedded systems. He has a B.S.E.E. from the University of Missouri-Columbia and an M.B.A. from Regis University in Denver. Stuart has authored three books about embedded systems. You may contact him at stuart@stuartball.com.

PROJECT FILES

To download the code, go to ftp.circuitcellar.com/pub/Circuit_Cellar/2005/178.

SOURCES

ATmega8515 Microcontroller
Atmel Corp.
www.atmel.com

DLP-USB245 USB Adapter
DLP Design
www.dlpdesign.com

FT245BM USB FIFO IC
Future Technology Devices International
www.ftdichip.com

Win32 Extensions
Mark Hammond
http://starship.python.net/crew/mhammond/win32/Downloads.html

Python scripting language
Python Software Foundation
www.python.org

The SBC One-Stop Shop
Panel PCs
PC Compatible SBCs
SoM & PC/104 Modules
Microcontroller SBCs A/D - D/A
Microprocessor Training Systems
Custom Engineering
Linux
2.4 Kernel
Turn-Key Solutions!
EMAC, Inc. has been Designing & Integrating Single Board Computers since 1985. We offer a comprehensive line of products and services for the Embedded Systems market.
Since 1985
OVER
20
YEARS OF SINGLE BOARD SOLUTIONS
EMAC, inc.
Phone 618 - 529 - 4525 Fax 618 - 457 - 0110
2390 EMAC Way, Carbondale, Illinois 62902
World Wide Web: http://www.emacinc.com

More Flash, Less Cash

Actel's flash memory FPGA concept has Tom excited. Read on to learn about the ProASIC3/E families of flash memory FPGAs.

It's no secret that I'm a big fan of FPGAs. My frequent articles on the subject are full of praise and bright hope for a future in which every designer has their own desktop foundry. I imagine simple development tools that make designing a System-on-a-Chip (SoC) as easy as using a paint program. Drag and drop an entire board's worth of modules you choose from a well-stocked library, click the Done button, and out pops your own homebrew custom chip.

These days you can even include a soft-core processor in your FPGA, with choices ranging from minimalist 8-bit controllers to full-blown, Linux/RTOS-capable, 32-bit processors. For the utmost in versatility and flexibility, you just can't beat soft software running on soft hardware.

When it comes to silicon, I admit I'm a hopeless optimist. But thanks to Moore's law, there's no folly in that. After all these years in high tech, I've learned that tomorrow's wunderchips are an ask-and-ye-shall-receive proposition. It's only a matter of when, not if. So, I'm sticking by my prediction that the future is bright for FPGAs and that they will play a big and growing role in mainstream applications.

PROMISE ALMOST SEEN

On the other hand…

The good news is that FPGAs have made great strides in overcoming the basic price and performance hurdles that have been their cross to bear since day one. While their silicon efficiency (i.e., area/gate) will never match a custom chip, keep in mind it doesn't have to. For many designs, the quick turnaround, reconfigurability, and versatility of an FPGA-based solution justify paying a bit more for each gate. That's especially true for applications in which other costs (design, support, and distribution) outweigh the silicon cost, which is after all the inexorable trend as long as costs keep rising for people and falling for silicon.

The FPGA price barrier may be gone, but that's not the end of the story. I'm reminded of the fall of the Berlin Wall, an event that inspired some people to speculate that it was the end of history and that we'd all live happily ever after. Yes, it was good news, but as we've seen, the fall of the Berlin Wall really just brought the next tier of challenges into view.

And so it is with FPGAs. For many applications, the raw chip price and performance are no longer barriers to design-in. But euphoria over that fact must be tempered with the recognition that a number of lesser, and maybe not so lesser, issues remain.

SRAM JAM

It's getting ludicrous! The latest FPGAs from programmable logic leaders Xilinx and Altera are looking more like memory chips in drag.

I'm not talking about the megabytes of block RAM that are accessible for your design. Block RAM is something that presumably is, or at least could be, useful in your application. Rather, I'm talking about the megabits, and even megabytes, of bench-sitting RAM that's used to configure the FPGA and otherwise serves no useful purpose (see Table 1).

The problems go well beyond the extra silicon and power wasted by the configuration RAM on the FPGA itself. Because the configuration RAM has to be initialized at power-up, that means you need another bunch of megabits (or megabytes) of memory to store a copy of the configuration, typically in an FPGA boot EEPROM. That's just great. Now you get to pay for, and power,

Virtex-4 platform FPGAs	Configuration bits	ISP PROM Solution
4VLX15	4,765,184	XCF08P
4VLX25	7,819,520	XCF08P
4VLX40	12,259,328	XCF16P
4VLX60	17,717,248	XCF32P
4VLX80	23,290,624	XCF32P
4VLX100	30,711,296	XCF32P
4VLX160	40,346,624	XCF32P*
4VLX200	48,722,432	XCF32P + XCF16P
4VFX12	4,765,184	XCF08P
4VFX20	7,242,240	XCF08P
4VFX40	13,550,336	XCF16P
4VFX60	21,002,496	XCF32P
4VFX100	33,065,024	XCF32P
4VFX140	47,856,512	XCF32P + XCF16P
4VSX25	9,147,264	XCF16P
4VSX35	13,699,904	XCF16P
4VSX55	22,744,832	XCF32P
*Assumes bitstream compression		

Table 1—*The latest SRAM-based FPGAs, such as the Xilinx Virtex-4, require many megabits of configuration bitstream and a big boot ROM (or two!) to hold them all.*

twice as much memory that sits around doing pretty much nothing. In an effort to counter the bloat, some FPGAs support compression of the bitstream.

But now the FPGA needs even more band-aids on-chip (decompression and decryption circuits) that, like the configuration memory itself, consume silicon and power to little useful end as far as your application is concerned. It's as though you were producing a Broadway play and decided to spend a bunch of your budget on a fancy way to raise the curtain.

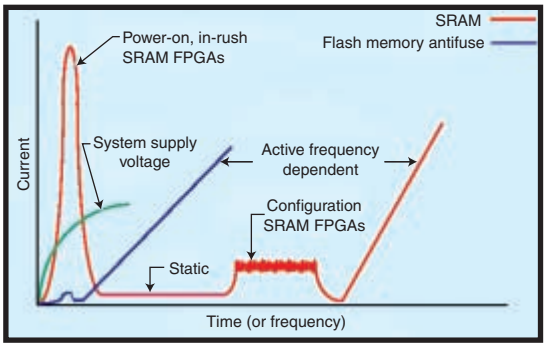


Figure 1—*The big power-on current surge of an SRAM-based FPGA is bad news for battery-driven applications. Furthermore, the configuration process itself (i.e., downloading the bitstream) consumes power and takes time. By contrast, flash memory-based FPGAs are live and less demanding at power-up.*[1]

FPGAs are already an iffy proposition for low-power (i.e., battery-driven) applications, and having to shuttle a bunch of configuration bits from here to there every power cycle doesn't help. But, in fact, the configuration process itself isn't the worst of the power woes.

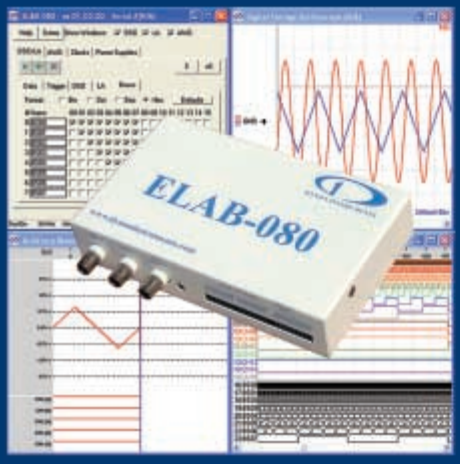
Consider that when power is first applied, the contents of the FPGA configuration RAM are in a random state. Imagine taking a bag full of ICs and wiring them up randomly. You don't have to be an electronics expert to anticipate that the resulting crossed connections and contending buses might be problematic. In fact, if you take a close look at the SRAM-based FPGA datasheet specs, or better yet, an oscilloscope trace like the one in Figure 1, you'll likely notice a relatively huge and long-lasting current surge at power-up.

This is obviously problematic, perhaps even a showstopper, for battery-driven applications. But even if power isn't a concern per se, it still means you have to size up your power supply circuits to accommodate the surge. Otherwise, voltage levels and rise times may fall out of spec and cause the FPGA to burp during start-up. And, of course, Murphy assures us that such glitches probably will be intermittent, the worst kind to deal with.

Then there's the small matter of cosmic rays. If it seems as though the stars are aligned against SRAM-based FPGAs, well, that's literally true. As I've mentioned in previous columns (most recently, "Hot Enough For You?" 174, January 2005), it's becoming increasingly clear that ever-finer geometry SRAM cells are susceptible to corruption from the cosmos. This is a blue-sky problem that isn't blue sky. The concern already has risen to a level that affects airborne and space-based applications. And the future for SRAM in this regard—a future where designers might have to worry about the altitude difference between Dallas

DYNON INSTRUMENTS

A Complete Electronics Lab for $495



Includes 5 Instruments
- 2 Channel 80 MHz Digital Storage Oscilloscope
- 16 Channel Logic Analyzer, Synchronous with DSO
- Arbitrary Waveform Generator
- 2 Programmable Power Supplies
- 2 Programmable Clocks

Plus....
- Powerful PC User Interface Program
- Probes
- Cables

Now includes FFT!

ELAB-080

www.dynoninstruments.com
Woodinville, WA 98072 (425) 402-0433

Embedded Solutions.

Dozens of Embedded Controllers with Countless Configurations to fit ANY Industry.

ATMEL AVR® Based Controller

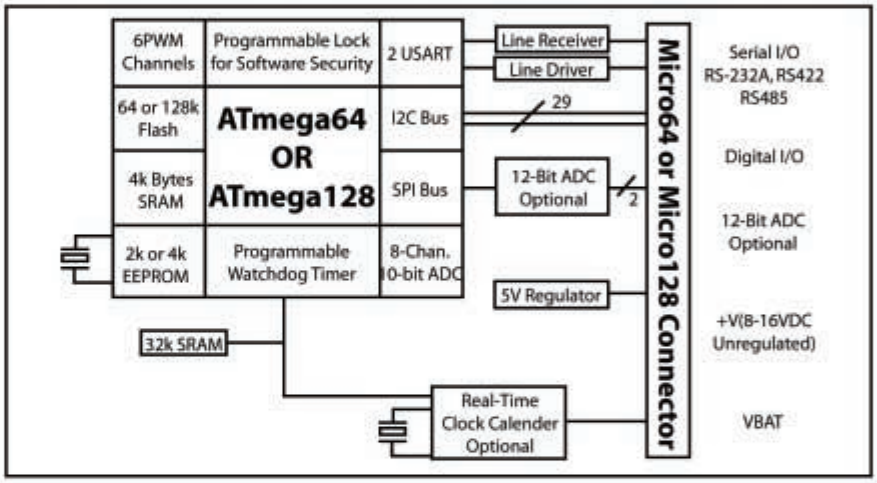
Micro64
AVR® Based Controller

■ Pin 1 www.micromint.com

- Small size - 1.5" x 2.1" x 0.52
- Dual powered - operates on +5V or 6.5-20V at 55 mA (typical)
- Program and Data Memories
  - 64k or 128k Bytes of In-System Reprogrammable Flash with 10,000 Write/Erase Cycles
  - 2 or 4 K Bytes EEPROM with 100,000 Write/Erase Cycles
  - 36K Bytes SRAM
  - Programming lock for Software Security
- Peripheral Features

Starting at $119 - Single Qty

- Optional 2-channel 12-bit ADC
- 8-channel 10-bit ADC
- 8 Single-ended Channels
- 7 Differential Channels
- 2 Differential Channels with Programmable Gain (1x, 10x, 200x)
- Byte-oriented Two-wire Serial Interface (I2C)
- Dual Programmable Serial USARTs
- 1 TTL
- 1 RS-232A, RS-422, or RS485
- Master/Slave SPI Serial Interface
- Two 8-bit Timer/Counters with Separate Prescalers and Compare Modes
- Two Expanded 16-bit Timer/Counter with Separate Prescaler, Compare Mode, and Capture Mode
- Two 8-bit PWM Channels 6 PWM Channels with Programmable Resolution from 1 to 16 Bits
- Programmable Watchdog Timer with On-chip Oscillator
- 29 Digital I/O that can sink or source 20 mA



Data sheets, Online Ordering and a Complete Catalog of Products at...

www.micromint.com

Delivering Embedded Controllers for 24 Years. 24 YEARS

Reduce Your Time to Market.

Minimize your time from conception to production by utilizing one of Micromint's market-proven controllers. Whether your concerns are digital or analog, inputs or outputs, Micromint has a product to fit your needs. Order quantities of one to thousands. Custom design and configurations are available.

With over 500,000 controllers in the marketplace, Micromint has been providing innovative, turn-key solutions to the OEM market for 24 years—from design through production, as well as packaging and shipping the final product. Our broad line of embedded controllers and turn-key solutions can turn your imagination into reality.

Visit our website @ www.micromint.com to see our complete line of OEM Solutions.

Micromint

115 Timberlachen Circle | Lake Mary, FL 32746 | 800-635-3355 | 407-262-0066 | Fax 407-262-0069

	A3P030	A3P060	A3P125	A3P250	A3P400	A3P600	A3P1000	A3PE600	A3PE1500	A3PE3000
System gates	30,000	60,000	125,000	250,000	400,000	600,000	1 million	600,000	1.5 million	3 million
VersaTiles (D flip-flops)	768	1,536	3,072	6,144	9,216	13,824	24,576	13,824	38,400	75,264
RAM	–	18 Kb	36 Kb	36 Kb	54 Kb	108 Kb	144 Kb	108 Kb	270 Kb	504 Kb
4,608-Bit blocks	–	4	8	8	12	24	32	24	60	112
FROM bits	1 Kb	1 Kb	1 Kb	1 Kb	1 Kb	1 Kb	1 Kb	1 Kb	1 Kb	1 Kb
Secure (AES) ISP	–	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Integrated PLL in CCCs	–	1	1	1	1	1	1	6	6	6
VersaNet Globals	6	18	18	18	18	18	18	18	18	18
I/O Banks	2	2	2	4	4	4	4	8	8	8
Maximum user I/O	81	96	133	157	194	227	288	270	439	616
Package pins QFN	QN132									
VQFP	VQ100	VQ100T	VQ100	VQ100						
TQFP		Q144	TQ144							
PQFP			PQ208	PQ208	PQ208	PQ208	PQ208	PQ208	PQ208	PQ208
FBGA		FG144	FG144	FG144, FG256	FG144, FG256, FG484	FG144, FG256, FG484	FG144, FG256, FG484	FG256, FG484	FG484, FG676	FG484, FG896

Table 2—*The new Actel ProASIC3/3E lineup brings the benefits of flash memory to applications covering the spectrum from 30,000 to 3 million gates and from 100 to nearly 1,000 pins.*

and Denver isn't bright.

Wasted configuration memory, strike 1. Power-up surge and delay, strike 2. Cosmic corruption, strike 3. SRAM-based FPGAs, you're outta there!

FLASH-O-MATIC

Although it's a tiny company compared to FPGA heavyweights Altera and Xilinx, I've got to say I really think Actel is onto something with their ProASIC flash-based FPGAs. This isn't news. Actually, the story began five years ago when Actel acquired flash memory-based FPGA technology from a company called Gatefield, supplementing their anti-fuse-based lineup.

I was optimistic about their flash memory-FPGA foray when I first covered it back in 2002 ("FPGA News Flash," 145, August 2002). With Actel's latest announcements of new chips, tools, and even a new marketing philosophy, I'm actually more fired up now (see Table 2).

Like the earlier ProASIC and ProASIC+ parts, the third-generation ProASIC3 (and ProASIC3E) family relies on a sea of VersaTiles that can each function as a three-input look-up table or a D flip-flop (see Figure 2). Such relatively simple logic elements form the basis for Actel's claim that, as the name implies, ProASIC is the most gate array-like FPGA, and thus a good fit with existing ASIC design tools and methodology.

But, as competitors are quick to note, simple logic elements require more interconnections and, thus, put pressure on routing resources. Actel says their seven-layer metal process implementing a four-level routing hierarchy avoids bottlenecks and achieves high gate utilization ratios.

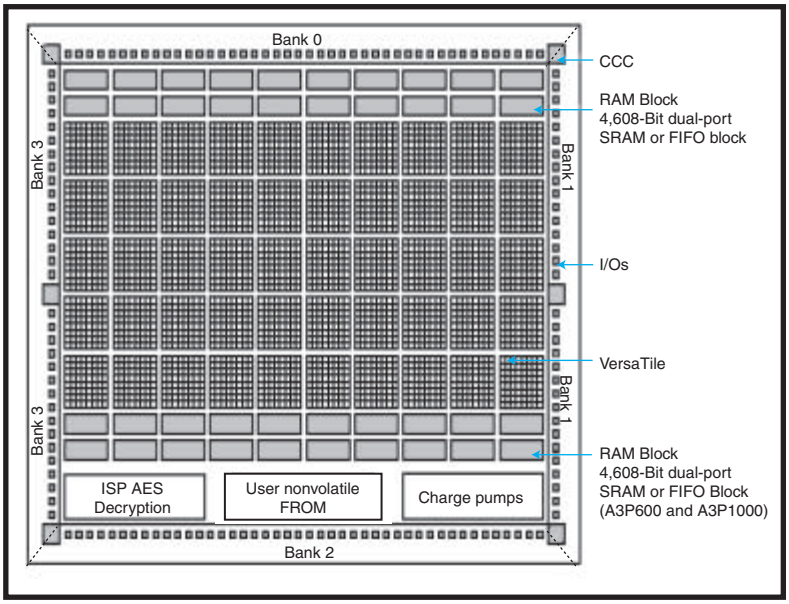


Figure 2—*Under the hood, the Actel ProASIC3 family features an array of VersaTiles surrounded by block RAM, multiple I/O banks (supporting different voltage levels), and clock-conditioning circuits (CCC). Bitstream decryption (not included on the entry-level A3P030), user FlashROM (FROM), and charge pumps (for single-voltage, in-system programming) round out the chip.*

In case you're wondering what the difference is between the ProASIC3 and ProASIC3E versions, the answer appears to be not much beyond some additional I/O standards and clock resources. Otherwise, ProASIC3 versus ProASIC3E seems to be a marketing scheme aimed at positioning the parts into low-end (less than 1 million gates, less than 200 pins) and high-end (greater than 1 million gates, greater than 200 pins) segments with some overlap in the middle. In particular, the Actel marketing department will get a lot of mileage out of the fact that every part in the ProASIC3 lineup comes in under $10 in high volume.

The entry-level A3P030 (30,000 gates) is a bit different from all the other parts. Actel is clearly positioning it as a loss leader for applications that don't need block RAM, bitstream encryption, or a PLL. At only $1.50 in high volume, their loss is your gain.

One standout feature that's unique to all the new ProASIC3 and ProASIC3E parts is the so-called FlashROM (FROM), which is a 1-Kb flash memory. Unlike the

Photo 1—*The board and programmer are nice, but it's the big-ticket software, including Synplicity and ModelSim, that makes the $249 ProASIC3 starter kit a real bargain.*

configuration flash memory, FROM is accessible to your application for miscellaneous data storage such as serial numbers, IP addresses, calibration factors, and so on. Interestingly, although the FROM is both read- and write-accessible via the JTAG programming link, it's only read-accessible from the FPGA logic, which explains the ROM label. There's probably an argument for preventing the FPGA logic from writing to the FROM (perhaps security or power distribution issues?), but the restriction does seem limiting.

TOOL TIME, AND TIME AGAIN…

Actel's flash memory FPGAs look good on paper, but what about the development tools? After all, tools are everything when it comes to getting a successful FPGA design out the door.

Actel will soon support the ProASIC3 line with a new starter kit (see Photo 1). Fortunately, the current ProASIC+ starter is similar, so I decided to check it out as a stand-in for the forthcoming ProASIC3 version. The bottom line is that the starter kits are a remarkable deal at only $249, including the board, PC plug-in JTAG downloader, and development software. Notably, you get the Gold version of Actel's Libero IDE that integrates heavy-hitter ($$$), third-party packages such as Synplicity (Synthesis) and Mentor Graphics's ModelSim.

As far as I can tell, the software doesn't expire or otherwise lack critical features as is common in other evaluation kits. The only gotcha is a limit in the capacity of the particular FPGA chips the package supports. The ProASIC+ kit I evaluated tops out at 300,000 gates (APA300 chip). I'm guessing the cap will be raised even higher for the forthcoming ProASIC3 kit because it's supposed to have a ProASIC3E chip installed, the smallest of which is 600,000 gates. That's a lot of gates, certainly plenty enough for evaluation and even development for a lot of full-fledged, real-world applications.

I congratulate Actel (and other FPGA suppliers too) for their continual efforts to craft better, and notably lower-cost, tools. But hold your horses. One thing I've learned over the years is that FPGA tools tend to be demanding, finicky beasts that must be coaxed into cooperation. In fact, as I recall, just about every time I want to experiment with FPGAs, I have to upgrade my PC hardware and software.

This time around I hoped my rather muscular PC would be up to the task. Not quite. The Actel installation guide noted that the latest versions of the tools are only happy with Windows NT, 2000, or XP Professional, but my PC had Windows XP Home. Normally, I would have attempted installation anyway, but I figured an XP Home-to-XP Pro upgrade would be relatively painless, so why not go for it?

At first I thought the installation was going well. The XP installer said it would take 51 minutes (!) and proceeded to churn away, so I took a break.

Checking in about 15 min. later, I noticed the system rebooting followed by a "Restarting Installer" message. Hmm, that seemed a bit odd, but so did taking 51 minutes. The Microsoft documentation said little about what was supposed to be happening at this point, so I hoped it was just a quirky installation procedure. But alas, while monitoring the situation, it soon became pretty clear that the installer was stuck in an infinite restart-reboot loop. Help! It wasn't long after that I was on the phone (at least on hold) to Microsoft.

When I finally got to a person, they were actually helpful, and I was able to solve the problem. With XP Pro up and running, I turned my attention to the software that came with the Actel kit.

Next time you're talking to an FPGA supplier tech support person, ask them if they've ever gotten any complaints about licensing. Chances are they'll take a well-deserved opportunity to vent. I've probably installed FPGA tools a dozen times over the years and licensing invariably seems to be a hassle.

No exception this time. To be fair, the main problem I encountered was because the kit was shipped to me outside normal channels. As a result, some of the licensing information fell into a crack along the way. No matter what I tried, I could only get the equivalent of a Libero Silver installation working, although the kit purportedly included Gold. Thanks to good support from Actel, I got past the problem ~~a~~ problem that shouldn't be an issue for normal purchases.

Not that there weren't a few head scratchers and potential pitfalls for the unwary. The documentation is insistent that you go to the Actel web site and register for a software license (sigh) during (not before or after) installation. Then you wait (the documentation says up to 2 h, but it was only 10 min.) to receive an e-mail with the license information, install the license, and finish the installation.

Sure enough, halfway through the installation, I was sent over to Actel's web site to register, which I did. I got the license, dinked with various settings (e.g., environment variables to point to the license file), and went back to finish

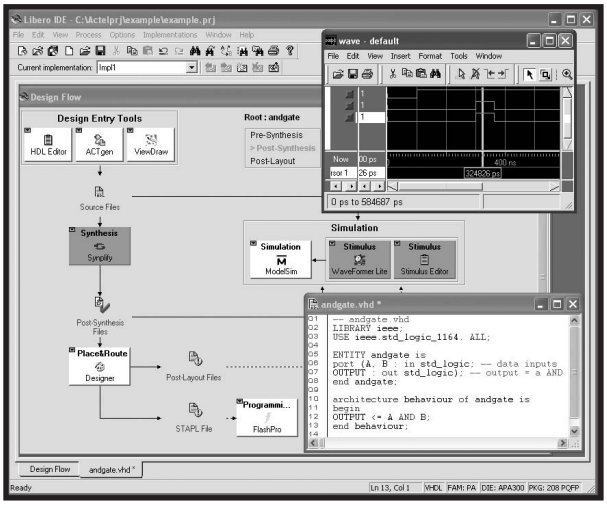


Photo 2—*The Libero IDE opens with a roadmap showing you the route to your own chip: enter the design, synthesize and simulate it, download, program, and repeat.*

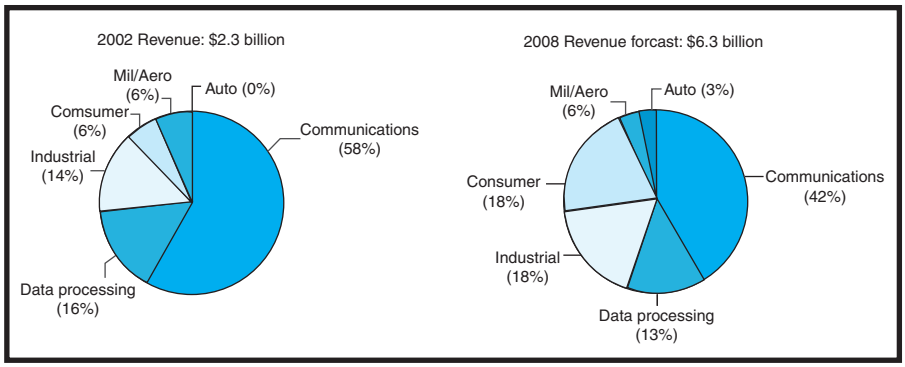


Figure 3—*So-called value-proposition FPGAs (i.e., low price and system cost) will play a big role in boosting FPGA sales, especially in consumer and industrial applications.*[2]

the installation. But where was the installer? The dialog box seemed to have disappeared in the interim. Poking around, at some point I closed the Explorer window I'd been using to register at the Actel web site. Suddenly, the installer dialog reappeared and I was able to complete the installation successfully.

Although most tools have moved on to USB (notably including the forthcoming ProASIC3 kit), the FlashPro Lite downloader/programmer that came with the ProASIC+ starter kit relied on a parallel port connection. Actually, because nobody uses parallel ports anymore, that means most of them, like mine, are gathering dust, so nothing needs to get unplugged. I gave the lash-up a 50/50 chance of working without fiddling around with a BIOS setting (e.g., EPP and ECP) or some such, but I was relieved to find I was able to connect and download successfully.

GATE DATE

I expect, or at least hope, few of you will encounter the upgrade, installation, and licensing hassles I went through. On the other hand, fair warning that such are not uncommon when it comes to the latest and greatest chip design software.

Anyway, with everything up and running, I was able to proceed through the tutorial, which walks you through the process of getting your ideas down on silicon. The Libero opening screen served as a good road map by showing each step along the way (see Photo 2).

First, you create a project that serves as a central repository for the many files (e.g., design, test, bitstream, etc.) that go into your design. You can mix and match HDL, schematics, macros, and canned IP.

After you have a design, the next task is functional simulation, using ModelSim to confirm the behavior of your design at a source code level. The first step is to create a test bench (the stimulus for simulation) by drawing input waveforms. Running the tutorial example (a simple AND gate) through ModelSim, you can confirm that setting both inputs high drives the output high.

Knowing that your design is correct in principle, it's time to proceed to see if it works in fact by synthesizing (Synplicity) it into an actual netlist. Following synthesis, you can simulate once again to confirm that the synthesized design still behaves properly.

The next step in the implementation is to have Libero place and route the netlist to target specific gates on a specific chip. Once again, you can simulate the design, this time taking advantage of actual timing information extracted (i.e., back-annotated) from the placed and routed implementation. Presuming all is well, the final step is to create the bitstream file and burn it into the FPGA.

Like riding a bike, crafting a chip is a slow and awkward process at first. I probably spent at least half an hour to get a single AND gate wiggling in the FPGA. It didn't help that some of the screens on the PC didn't quite match those shown in the printed tutorial documentation. I'll cut Actel some slack because it's practically inconceivable to expect printed documents to stay in sync with such a complex ever-evolving conglomeration of software. There's even a typo in the tutorial VHDL file itself. Maybe it's there on purpose so you can practice the edit-recompile cycle that you'll ultimately become extremely familiar with.

Again, like riding a bike, practice will no doubt make things easier. Notably, as you get up to speed you can tweak various settings (e.g., default window arrangements and command scripts) to streamline things and have it your way.

PROPOSITION ME

I believe Actel is doing right by the flash memory FPGA concept. Beyond the chips and tools themselves, it's clear they are aggressively targeting high-volume mainstream applications, the so-called "value proposition" market. [1] That's market-speak for "cut the price and sell a heck of a lot more chips" (see Figure 3).

I can even see these flash memory FPGAs enabling a new generation of entrepreneurial chip designers. Just as the emergence of IC foundries allowed so-called "fables" IC companies to flourish, I can see garage shops using Actel chips as the basis for their own foundry-less startups.

I do hope for continued progress making the tools easier to use and more accessible to blue collar designers, not just chip design gurus. I expect that will happen. The sands of time, and cries for tech support, will inexorably wear down the rough edges.

If flash FPGAs are the answer, how Xilinx and Altera will respond is an interesting question. Don't you just love this business?

Tom Cantrell has been working on chip, board, and systems design and marketing for several years. You may reach him by e-mail at tom.cantrell@circuitcellar.com.

REFERENCES

[1] Actel Corp., "Understanding FPGA and Total System Power," April 2003, www.actel.com/products/rescenter/power/docs/System_Power.pdf.

[2] ——, "Flash FPGAs in the Value-based Market," January 2005, www.actel.com/documents/ValueFPGA_WP.pdf.

SOURCE

ProASIC Flash memory-based FPGAs
Actel Corp.
www.actel.com

IDEA BOX

THE DIRECTORY OF PRODUCTS AND SERVICES

AD FORMAT: Advertisers must furnish digital submission sheet and digital files that meet the specifications on the digital submission sheet. **ALL TEXT AND OTHER ELEMENTS MUST FIT WITHIN A 2″× 3″ FORMAT**. Call for current rate and deadline information. Send your disk and digital submission sheet to: IDEA BOX, Circuit Cellar, 4 Park Street, Vernon, CT 06066 or e-mail kc@circuitcellar.com. For more information call Sean Donnelly at (860) 872-3064.

The Suppliers Directory at www.circuitcellar.com/suppliers_dir/
is your guide to a variety of engineering products and services.

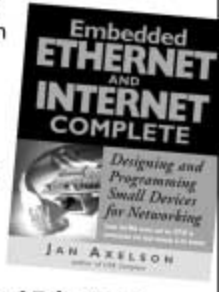
PHYTEC
phyCORE® OEMable Single Board Computers
8051 CAN
C166/XC16x/ST10
PowerPC
x86: Flash SC520
ColdFire
XScale PXA255
ARM7 AT91SAM9260
[PQ2FB]
- sub-miniature SBC subassemblies (as small as 47x55 mm)
- low EMI, multi-layer design plugs like "big chip" into target circuitry
- 32 KB to 128 MB external SRAM and Flash (controller-dependent)
- CAN, Ethernet, RS-232/485, USB, D/A; A/D, DAC (controller-dependent); free I²C® and SPI
- available in Rapid Development Kits w/demos and compilers
- flexible BSP: insert-ready COTs subassemblies accompany you from evaluation through prototyping to end production... accelerating your design cycle and reducing time-to-market

www.phytec.com ■ (800) 278-9913 ■ www.phycore.com

video|text overlay
Turn serial data into video text:
BOB-3 · XBOB · COG-4
www.decadenet.com
DECADE ENGINEERING
503-743-3194 Turner, OR, USA

I2C, MDIO, CAN, SPI, XA Kits, & ISP FLASH
PC-ISP-FLASH programs 80C51, XA and AVR ISP/IAP FLASH micros from Philips and Atmel with user upgradeable Windows software. ISP interface cable, software and documentation included for **$199.00**.
PC-I2C-KIT allows bi-directional communications between any PC and I2C, SPI or MDIO based peripherals. Windows software, board, interface cable and documentation included for **$249.00**.
PC-I2C-DEV lets the user write programs to control the I2C/SPI/MDIO interface board directly using Microsoft Visual C/C++, Borland C/C++, Microsoft Visual Studio, etc. Includes a software DLL for custom user applications and a complete PC-I2C-KIT for **$499.00**.
XTEND Kits allow the user to debug or prototype any Philips XA based design. Kits include XA Dev Board with Monitor and Debugger, Software, Documentation, Schematics, Power Supply and Serial Cable.
XTEND-Gxx (G3 & G49) or **XTEND-Sxx** (S3) for **$249.00**
XTEND-C3 or **XTEND-H4** for **$299.00**
FDI
www.teamfdi.com
VISA/MC/Amex
Future Designs, Inc.
2702 Triana Blvd
Huntsville, AL 35805
(256) 883-1240
Fax (256) 883-1241

USB
Add USB to your next project—it's easier than you might think!
USB-FIFO up to 8 mbps
USB-UART up to 3 mbps
USB/Microcontroller boards
pre-programmed with firmware
Absolutely NO driver software development required!
www.dlpdesign.com
Design services available
DLP Design

Put Your Devices on the Internet
(and in local networks too)
Networking isn't just for PCs. Learn how to design and program small devices that exchange data via TCP/IP, host Web pages, communicate using e-mail and FTP, and much more!


Embedded Ethernet and Internet Complete
by Jan Axelson
ISBN 1-931448-00-0 $49.95
Lakeview Research www.Lvr.com
From the author of *USB Complete*

Serial, Ethernet, Web, Email
Simple, DTE, DCE, No SW Changes
$99.95 Qty 1

Grid Connect Inc.
+1 800 975-GRID
www.serial-to-ethernet.com

DEVICE PROGRAMMER, USB ROM Emulator, EPROM ERASER

TopMax **ChipMax**

$995 $345

TopMax : Programs over 7000 devices, 48 Pin ZIF Socket ,Tests TTL/CMOS Logic IC, Low voltage, On-board processor and built-in Universal AC, 9X/NT/2000/XP,CSA, UL, Made in USA, Lifetime free software, 2 year warranty, Free software update for life time

TM-8G - DIP	8-gang (32-pin DIP) adapter for E(E)PROM	$325
TM-8G - PL	8-gang (32-pin PLCC) adapter for E(E)PROM	$495

ChipMax : Reliable and low price ,Programs over 4000 devices, EPROM, FLASH Memory, Microcontrollers, PIC, Serial EPROM, EPLD, GAL, PALCE, 40 Pin ZIF Socket, Parallel, 9X/NT/2000/XP, Made in USA ,1 year warranty, Lifetime free software, 110 or 240 AC Adapter. CM-4G: 4-gang adapter 32-DIP $115

EeRom-8U

$495

"USB Interface"

EeRom-8U : 8Mbit-45ns EPROM/Flash Emulator, High speedy USB port interface (Plug & Play)

- Fast download (8Mbit per second), Lower Voltage, 8bit, 16bit, and 32bit data split
- High Speed access time (built-in 8M SRAM 15ns)
- No external power is needed, 9x/XP/NT/2000
- Accepts all standard Hex formats and binary files
- Jumper less Emulation size configuration

LA6T-110

$115

LA6T-110 : UV ERASER for Personal user, Fast and efficient, Molded Plastic Case, Hold 9 of 28-pin or 6 of 40-pin EPROM, Timer(15-40 min),
Programming Adapters(low price) : 8-SOIC($70), 32-PLCC($70), 44-PLCC($115), 48-TSOP($135), 44-QFP($65), 52-PLCC($165), 68-PLCC($165) ,
EPROM Emulator Adapter, 32/28-pin($95) 44-pin($115) DIP-to-PLCC

EE TOOLS

E.E. Tools, Inc.
549 Weddell Drive
Sunnyvale, CA 94089
www.eetools.com

Tel. 408-734-8184
Fax. 408-734-8185

VISA MasterCard

IDE, Compilers, ICD , Simulators, Programmers...

Emulators

8051 PIC® AVR®
80196 XEMICS
MSP-430 SENSORY

Phyton 718.259.3191
www.phyton.com

Custom Printed Circuit Boards

Nobody beats our PROTOTYPE PCBs
We are the 5 PCBs/5days specialist

AMAZING VALUE

2 layer: Get 5 PCBs for $13 each in 5 days
4 layer: Get 5 PCBs for $33 each in 5 days

(Upto 20 sq inch, FR-4, 0.062" thick, 7 mils trace/space,
15 mils finished hole min., unlimited drill sizes,
FREE Tooling, green LPI soldermask, white silkscreen)

(More sizes and offers available. See website for details)

Order Online
www.PCBFabExpress.com
Phone: (408) 857 0039
email: support@pcbfabexpress.com

PCBFABEXPRESS
High Quality PCBs @ Low Impact Prices

ElectronicKits.com

Usb/Serial Port PIC Programmer
Educational Electronic Kit $34.95

* Connect to computer via USB or Serial Port
* Free Software and Updates
* Windows 9x/Nt/2000 XP Compatible
* ICSP (In Circuit Serial Programming)
* Quality PCB and Silkscreen

See complete details at www.ElectronicKits.com

USB and USB Flash Pic Programmers Available
Carl's Electronics Inc. Offering hundreds of Educational Hobby Kits
Visit our website at www.ElectronicKits.com

NanoCore12™ for best price & performance: 9S12C microcontroller modules from $45!

NanoCore12™ Features:

- on-board RS232C interface
- up to 33 I/O lines, with multi-property programmability (e.g. direction, pull-up/pull-down, reduced drive, invert polarity, etc.)
- up to 8 key wake-up interrupt inputs, with digital filtering
- SCI • SPI • CAN
- 8-channel 10-bit ADCs
- multiple PWM channels
- 4-channel timers, supporting input capture/output compare, event counting, gated time accumulation, and simple PWM
- 32K multi-sector Flash • 2K RAM
- operates up to 48MHz via PLL
- 3.3 V or 5 V operation
- advanced CISC architecture
- on-chip Serial Monitor
- supports BDM debugging

1-877-963-8996 • www.technologicalarts.com
Toll-free in USA and Canada

™NanoCore12 is a trademark of Technological Arts, Inc.

Get the most out of the Rabbit's powerful feature set with the first reference book dedicated exclusively to the Rabbit 3000.

Embedded Systems Design Using the Rabbit 3000® Microprocessor

By Kamal Hyder and Bob Perrin

Buy Online $49

RABBIT Semiconductor

www.rabbit3000book.com

SHIPPED NEXT BUSINESS DAY FILE IN before 1PM EASTERN
$85 US
w/ solder mask
no drill size restriction
5" x 6"
2-layer - 2 pc
PROTOTYPE
Printed Circuit Board
Also available
Small Run Production
5 days turn
1pcb.com

USB Serial & Digital
USB to Serial
1, 2, 4, and 8-port models
RS-232, RS-422, and RS-485 interfaces
Data rates to 921.6K bps
USB to Digital
48/96 TTL (SSR rack compatible)
Reed and Form C relay outputs
Optically isolated inputs
Daisy chain multiple units
SEALEVEL
Sealevel Systems, Inc. 864.843.4343 TEL
sales@sealevel.com www.sealevel.com

okwelectronics.com
In-Circuit Emulator
ICEPIC 3
Next Generation Emulation!
All new universal In-circuit emulator for the PIC Microcontroller. Exceptional performance and value.
FEATURES
• Operates within MPLAB
• Hardware support for single steps in 'C'
• Fast USB connection
• Microchip assembler and 'C' compatible
• Source level debug in assembler or 'C'
• Full speed 40MHz emulation
From only $900*
• 2.8/5V Emulation
• Plug-in daughterboards ordered separately to configure system to the desired PIC
*Daughterboard not included in price
SEE OUR WEBSITE FOR PIC COMPATABILTY!
More info. @ www.okwelectronics.com
OKW Electronics, Inc.
Tel: (800) 965 9872 Fax: (412) 220 9247
300 Old Pond Road, #203, Bridgeville, PA 15017
Email: sales@okwelectronics.com

okwelectronics.com
Radio Modules
AM & FM
Off-the-shelf solutions
MANY NEW MODELS!
From $6.00 for 100 pcs
• Low cost AM & FM transmitters and receivers •
• High performance FM modules •
• Narrow band FM modules •
• Transceivers and Antennas •
• 315MHz, 418MHz & 433.92MHz •
• Range up to 1900 feet •
More info. @ www.okwelectronics.com
OKW Electronics, Inc.
Tel: (800) 965 9872 Fax: (412) 220 9247
300 Old Pond Road, #203, Bridgeville, PA 15017
Email: sales@okwelectronics.com

NavDrive™ II
Autonomous Robot Controller
NavDrive is a combination of software and hardware for building custom mobile robot applications. Source code is provided for the extendable C++ framework that integrates sensor data, and controls a robot's path while avoiding objects. On-board flash contains a complete application demonstrating NavDrive's capabilities and tools for installing custom applications.
• 5"x8"
• PowerPC™
• 3 Axis Compass
• 3 Axis Accelerometer
• 3 Axis Gyroscope • GPS Interface
• 4 Channel Ultrasonic • 2 Encoder Inputs
• 2 Axis Motor Control - Tracked or Steered
• Many Digital/Analog Inputs and Outputs
Integrated Knowledge Systems
www.IKnowSystems.com
602-705-8978

Add a color touch interface to your embedded product!
• High level RS232 interface
• Low cost
• In stock!
• Easy to program
Add color graphics to any 8/16 bit embedded system.
Easy, fast and flexible. Up and running in hours!
REACH TECHNOLOGY INC.
www.reachtech.com • 510-770-1417
842 Boggs Avenue • Fremont, CA 94539

LPC2000 Philips ARM Design Contest 2005
Enter Now!
Submission deadline:
October 24, 2005
More Circuit Cellar Contest Excitement!
Wireless Design Challenge.
Featuring ZigBee™ Technology
freescale semiconductor
Thank you to everyone who participated in the Wireless Design Challenge.
The entries are now being judged.
Stay tuned for the winners announcement.
www.circuitcellar.com/magazine/contests.htm

INSTANT MARKETING!

Do you have a great product but you don't want to bother with all the marketing, phone calls, and order fulfillment? We can handle product promotion, sales and invoicing, credit card services and shopping cart.

- Free webpages!
- Fulltime phone response!
- Instant sales office!
- Get more time for yourself for designing new products!

We can shoot product photos, produce literature, issue press releases and create a market for your product.

Saelig Company Inc
Email: promotion@saelig.com
Call: 1-888-7SAELIG for details.

ANSI C software development tools for:
IMAGEcraft™
NEW: ARM7 compiler (Philips LPC, Atmel SAM, etc...)
Version 7 Tools Full-featured 45-day demos on our website!
CypressM PSoC
AVR & tinyAVR
HC08,11,12,16
TI MSP430
ARM
hardware kits:
Elektronikladen - HC08 & HCS12 / 12 ref. design & starter kits
CANDIP - AVR/CAN on a chip!
Ethernut - Complete AVR TCP/IP solution
high powered development, garage powered prices!
www.imagecraft.com
info@imagecraft.com
706 Colorado Ave. Palo Alto, CA 94303
(650) 493-9326 • FAX (650) 493-9329
VISA / MC / AMEX
Check / Money Order
...say 'NO' to JURASSIC PRICES!

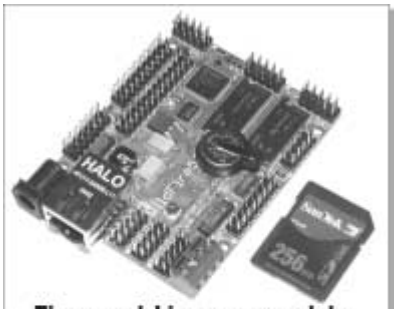
PROTOCESSOR
Add Industrial Automation Protocols to your Device - Fast
Modbus/TCP
EtherNet/IP
Modbus/RTU
SNMP
HTTP
Webserver
BACnet
Ethernet, RS-232 & RS-485
Metasys
Ensure connectivity using interchangeable, low cost OEM Modules with a common PCB footprint and software interface. Development kit with "C" source code available.
www.ProtoCessor.com
800-317-8319

I²C™/SMBus
- Bus Monitors
- Protocol Analyzers
- Host Adapters
- Multiplexers
- Battery Applications
- Software Tools
MCC
Micro Computer Control
I²C is a trademark of Philips Corporation
www.mcc-us.com

EmbedDisk -40 to +85°C Extreme Temp
32MB
IDE Flash Industrial Grade Storage
MSTI Media Stream Technologies Inc.
www.embeddisk.com
info@embeddisk.com
iCOP ICOP Technology Inc.
www.icoptech.com
info@icoptech.com

SmartTFT™ Ultra Bright, Wide Viewing Angle, Impressive Color Images on TFT
$99 OEM Boards
- 6.5 x 4.5 inches, Program in C/C++
- Relays, Opto-couplers, Solenoid, Quad. Decoders
- 16-bit ADCs and DACs, 4-20 mA
- Supports QVGA TFT, Touch sreen
- 10 BaseT Ethernet with TCP/IP
- CompactFlash with file system
- 160 field Removable industrial screw terminals
50+ Low Cost Controllers with ADC, DAC, solenoid drivers, relays, PC-104, CompactFlash, LCD, DSP motion control, 10 UARTs, 300 I/Os. Custom board design. Save time and money.
TERN INC.
1724 Picasso Ave., Suite A, Davis, CA 95616 USA
Tel: 530-758-0180 • Fax: 530-758-0181
www.tern.com • sales@tern.com

okwelectronics.com
Radio Modules
AM & FM
Off-the-shelf solutions
MANY NEW MODELS!
From $6.00 for 100 pcs
• Low cost AM & FM transmitters and receivers •
• High performance FM modules •
• Narrow band FM modules •
• Transceivers and Antennas •
• 315MHz, 418MHz & 433.92MHz •
• Range up to 1900 feet •
More info. @ www.okwelectronics.com
OKW Electronics, Inc.
Tel: (800) 965 9872 Fax: (412) 220 9247
300 Old Pond Road, #203, Bridgeville, PA 15017
Email: sales@okwelectronics.com



Tiny, yet Linux capable

16MB fast SDRAM
64MHz Coldfire MCU
1 - 4 MB flash memory
SD card socket (to 1GB)

10/100 Ethernet
3 serial ports (RS232)
1 CAN port
LCD/KPD port

Eclipse/CDT dev. env.

serial debugger incl.

64 I/O pins & capabilities to burn...

www.steroidmicros.com

Development Tools

PRESTO - USB Programmer

Extremely fast and flexible in-system programmer for Microchip PIC & dsPIC, Atmel AVR & 51, serial EEPROM & Flash, JTAG (CPLDs, FPGA PROMs), ...
Tired of waiting while programming your favorite chips? Buy PRESTO now!

MU Beta - PIC Emulator

Real-Time In-Circuit Emulator for popular PICmicro MCUs in 6 to 18-pin packages.
Unique, flexible and affordable - lowest cost per emulated device in industry!

And many other products...

ASIX **ASIX** (Czech Republic, EU)
info@asix-tools.com

www.asix-tools.com

China Connection...

PCB Production,
Assembly,
Turnkey

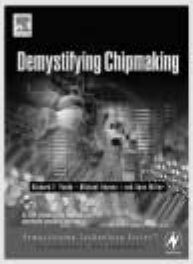
Prototype ~ Production

D/S PCB ~ 12¢ per in^2
4-L PCB ~ 20¢ per in^2

3 weeks Delivery w/solder Mask
NRE not included
UL Approved

MYLYDIA INC.
1-800-Mylydia
sales@mylydia.com
www.mylydia.com

10% Off &
Free Shipping



Takes the reader through the actual manufacturing process of making a typical chip, from start to finish!

Includes a detailed discussion of each chip making step, in plain language.

Newnes

Save 10% off and Free Shipping at
www.books.elsevier.com/computereng
Use Offer Code 81507

It writes your USB code!

NO Need to be a USB expert!

HIDmaker ($399) – creates ready to compile PC & PIC programs that talk to each other over USB.

Choose your favorite languages!
PIC: Pic Basic Pro, CCS C, Hi-Tech C, MPASM. **PC:** VB6, Delphi, C++ Builder.
Single chip solution: PIC with built-in USB

HIDmaker Test Suite ($149)

USBWatch – shows your device's USB traffic, even during 'enumeration', without expensive equipment.

AnyHID – Test any USB HID device. See what data it sends, even what the data is used for.

Trace SYSTEMS, Inc.
301-262-0300
WWW.TraceSystemsInc.com

PROBLEM SOLVED!

Custom Hardware
&
Software Design

- Rapid Prototypes
- Cost-effective Design
- Free Consultation and Quotes

MJS CONSULTING INC.
Mark J. Samarin M.S.E.E.
http://www.mjsconsulting.com
253-858-6844

www.can232.com

The original (Controller Area Network) CAN to RS232 serial converter.

Powered with an Atmel® AVR® RISC microcontroller and a Philips SJA1000 CAN controller makes a unique combination of high speed transfers possible even at higher CAN bus speeds. CAN232 has also a built in receive FIFO of 32 messages which makes it possible to receive small bursts of messages without losing any frames.

Does it sound interesting?
Read more at **www.can232.com**

* Free sample programs in Delphi, VB, Linux, C & C++
* 8-15VDC supply via CAN connector (CiA DS102)
* Small size 68mm by 30mm (2.7" by 1.2")
* Timestamp in mS on CAN frames
* Operating system independent
* Both 11 & 29 bit ID support
* ASCII or Binary protocol
* Firmware upgradable
* Custom design
* CE approved

Philips is a registered trademark of Koninklijke Philips N.V.
Atmel® and AVR® are registered trademarks of Atmel Corporation

"Your Source for Design & Mfg. Services"

- ✓Schematic Capture, PCB Design
- ✓Embedded Microcontroller Design
- ✓Hardware, Firmware Design
- ✓Rapid Prototypes, Fab and Assembly
- ✓RT-OS • C • C++ • ASM • Basic
- ✓PowerPCB • OrCAD • Allegro

Reliable ♦ Fast Work ♦ Free Quotes ♦ Guarantee!

VNISOURCE DESIGN
www.vnisource.com
Tel: (469) 675-0267
Fax: (469) 675-0346

PC KEYBOARD EMULATION

Interface Keypads, Switches, or RS-232 to your PC Keyboard Input

MODEL KE24 ONLY $99.95

• Up to 12 x 12 matrix • Programmable • RS-232 Port • Macro Capability

The KE24 is the ultimate in flexibility. Inputs or serial data can emulate any of the 101 keys from a standard keyboard.

MODEL KE18 ONLY $44.95

• Up to 9 x 9 matrix • 2.5" x 3.0" Size • PC Keyboard Port • PCXT, AT Compatible

The KE18 combines a multitude of features with small size at an economical price. Custom units available.

HAGSTROM ELECTRONICS
11 Fiddlers Green
Lansing, NY 14882

Toll Free: 888-690-9080
Phone: 607-533-4441 Fax: 607-533-4443
www.hagstromelectronics.com

VISA

ALL ELECTRONICS CORPORATION

Electronic and Electro-mechanical Devices, Parts and Supplies.
Wall Transformers, Alarms, Fuses, Relays, Opto Electronics, Knobs, Video Accessories, Sirens, Solder Accessories, Motors, Heat Sinks, Terminal Strips, L.E.D.S., Displays, Fans, Solar Cells, Buzzers, Batteries, Magnets, Cameras, Panel Meters, Switches, Speakers, Peltier Devices, and much more....

www.allelectronics.com
Free 96 page catalog
1-800-826-5432

PLCs for OEMs

Program like a PLC (Ladder + BASIC)
Low Cost & Flexible like a Micro-p

Ethernet XServer / Web HMI

$199

8 Analog i/os
24V Digital i/os
MODBUS
LCD Port, PWMs
RTC, Stepper
Interrupts,
Hi-Speed Counter
Modem / Internet

New Option:
Ethernet / Wi-Fi
w/ Free Web HMI

Incredibly Easy to Program!
Our software is used by many colleges for teaching PLCs!

Get Free Ladder Logic Simulator:
www.tri-plc.com/cci.htm

Triangle Research

Tel: 1-877-874-7527 - PLC specialist since 1993

DEVELOPMENT BOARDS

NEW USB Development Board

ONLY $32.90

- AT89C5131 Microcontroller
- 32kBytes Flash Memory
- Real Time Clock
- External EEPROM
- Easy USB Interface and Control
- In-Circuit USB Programming

Also Available: Opto-Isolated Relay Output Board, Web Control Board and Heaps More

www.futurlec.com

Got Dial Tone?

Telecom Hardware/Software Developers

STOP using your phone lines to test and demonstrate your telecom devices. Our affordable telephone line simulators offer authentic USA dial tone, busy signals and ringing. Supports high speed analog modems too!

RING-IT! TELCO SIMULATOR
- Caller-ID
- LED display
- Audio Output Jack
- Real 20Hz Ring
- ~~$325~~ Sale $299.95

PARTY-LINE TELCO SIMULATOR
- Six Extensions
- Caller-ID
- Distinctive Ringing
- CPC Disconnect
- ~~$425~~ Sale $399

Digital Products
COMPANY
134 Windstar Circle
Folsom, CA 95630 USA
Tel: 916-985-7219
Fax: 916-985-8460

VISA

http://www.digitalproductsco.com

QScreen™ - Low Cost Versatile Instrument Controller

$359/100s

- **Touchscreen Operated GUI**
- **128x240 CCFL- Backlit Graphic Display**
- **Programmable in C or Forth**
- **Hundreds of Screens, Buttons, & Menus**
- **A/D and Two RS232/485 ports**
- **8 Timer-Controlled Digital I/O Lines**
- **Up to 1 MB Flash, 512KB RAM**
- **Wide Selection of Plug-in I/O**

Mosaic Industries Inc.
tel: 510-790-1255 fax: 510-790-0925
www.mosaic-industries.com

ValueCAN

The High Value Tool For Controller Area Network

- USB to CAN
- Simple software analyzer included
- DLL with examples for custom applications
- PC isolated from CAN
- 100% bandwidth at 500Kb

Only

$295

ICS RADDAQ

www.intrepidcs.com/vcan

LEMONSTUDIOS

IN COOPERATION WITH

PARALLAX INC.

PRESENT...

INTRO TO ROBOTICS

If you want to get started in robotics but don't know where to begin, then this DVD is for you!

Using the most popular programmable robot in the world, the Parallax Boe-Bot, we show you how easy it is to learn about robotics.

WATCH THE TRAILER AND BUY YOUR COPY AT

WWW.LEMONSTUDIOS.COM
OR CALL
503-761-0964

Flashlite 186

- 186 processor @ 33 MHz
- DOS w/ Flash File system
- 44 Digital I/O lines w/ CPLD
- Console / Debug Serial Port
- 7-34V DC or 5V DC power
- 2 Serial Ports
- Accepts 8MB DiskOnChip
- 2 16-bit Timers
- 512K DRAM & 512K Flash
- Watchdog Timer
- Expansion options with Peripheral Boards

$69 QTY 1

$99 Development System

Development kit includes:
- Flashlite 186 controller
- Borland C/C++ ver 4.52
- FREE Email Tech Support
- Serial Driver library
- AC Adapter and cable

Call 530-297-6073 Email sales@jkmicro.com
On the web at www.jkmicro.com

JK microsystems

WWW.CONITEC.NET

$333 COMPLETE KIT

GALEP-4 UNIVERSAL PROGRAMMER

Introducing a pocket programmer with true **Universal Output**

Latest generation Galep-4 uses ASIC universal pin technology for each pin of 40 pin ZIF-socket. 5200+ device library / lifetime free updates. Programs 8/16-bit EPROM'S, EEPROMs, 0-Power RAM, FLASH, Serial EEPROM's, GAL, PALCE, microcontrollers such as 87/89xxx, PIC, AVR, ST62, etc. Low voltage devices down to 1.3V. No adapter required for DIL devices. 8 Hrs. operation on battery (AC charger included). Runs WIN 98,NT,ME, 2000,XP with Hex/Fuse Editor.

Remote control from other apps, (e.g. VisualBasic). Substitutes high priced universal programmers e.g. ALL-11 (HILO) or LAB-T00L-48 (ADVANTECH) Providing virtually matching performance at only 1/3-1/5 the price. Info, orders, softwr : 619-702-4420

ISO 9001 CERTIFIED 9001 PROFESSIONAL TOOLS

info, downloads: www.conitec.net

CONITEC DATASYSTEMS - 1951 4TH AVE, SUITE 301 - SAN DIEGO, CA 92101 - TEL: 619-702-4420 FAX: 619-702-4419

PROTOTYPE CIRCUIT BOARDS

$85 US for two 5" x 6" 2-layer boards

Shipped NEXT BUSINESS DAY if data is received by 1:00 pm EASTERN

www.apcircuits.com

AP Circuits

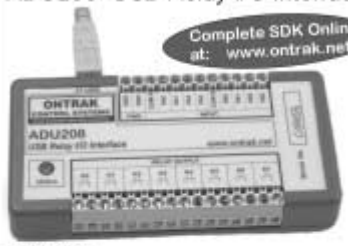
(403) 250-3406

staff@apcircuits.com

USB Data Acquisition

ADU208 -USB Relay I/O Interface

Complete SDK Online at: www.ontrak.net



FEATURES
-8, 5-AMP relay outputs
-8, ISOLATED digital inputs
-Port Powered, Aux 5VDC output $189.00 QTY 1

Other Models:
ADU200- 4 Channel Version with RS232 $139.00
ADR218- Solid-State Version 8 Channel $225.00
ADU100- 3 CH, 16-Bit ISOLATED Analog Inputs, PGA, 4 digital I/O, RS232 and 5 AMP Relay Output $199.00

ONTRAK CONTROL SYSTEMS INC.
PH: (705) 671-2652 Fax: (705) 671-6127

www.ontrak.net

The Pocket Programmer $149.95

The portable programmer that uses the printer port instead of a internal card.

Easy to use software that programs E(E)prom, Flash & Dallas Ram. 25/27/28 & 29 series from 16K to 8 Megabit. Adapters available for 874X, 875X MCU's, Pic, Serial Eprom's, 40-Pin X 16, Proms and More.

Same Name, Address & Phone # for 19 Years... Isn't it Amazing ?

Intronics, Inc.
Box 13723 / 612 Newton St.
Edwardsville, KS 66113 Add $7.00 COD
Tel. (913) 422-2094 Add $6.00 Shipping

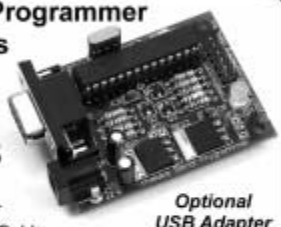
WWW.IN-KS.COM Visa / MC/Amex/Disc

Order online at: www.melabs.com

microEngineering Labs, Inc.
Development Tools for PIC Microcontrollers

Phone: (719) 520-5323
Fax: (719) 520-1867
Box 60039
Colorado Springs, CO 80960

Serial Programmer for PICs



$119.95
Includes:
Programmer
9-pin Serial Cable
AC Power Adapter
ZIF Adapter for 8 to 40-pin DIP
Software for Windows 98/Me/NT/2K/XP

Optional USB Adapter $39.95

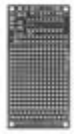
EPIC Parallel Port Programmer starting at $59.95

LAB-X Experimenter Boards

Pre-Assembled Board
Available for 8, 14, 18, 28, and 40-pin PIC MCUs
2-line, 20-char LCD Module
9-pin Serial Port
Sample Programs
Full Schematic Diagram

Pricing from $69.95 to $349.95

PICProto Prototyping Boards



Double-Sided with Plate-Thru Holes
Circuitry for Power Supply and Clock
Large Prototype Area
Boards Available for most PIC MCUs
Documentation and Schematic

Pricing from $8.95 to $19.95

BASIC Compilers for PIC MCUs

Easy-To-Use BASIC Commands
Windows 9x/Me/2K/XP Interface

PicBasic Compiler $99.95
BASIC Stamp 1 Compatible
Supports most 14-bit Core PICs
Built-In Serial Comm Commands

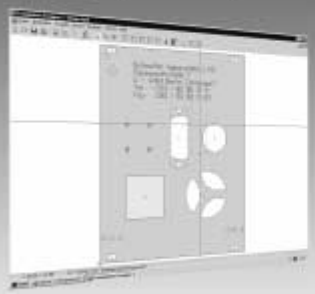
PicBasic Pro Compiler $249.95
Supports all PICmicro MCUs
Direct Access to Internal Registers
Supports In-Line Assembly Language
Interrupts in PicBasic and Assembly
Built-In USB, I2C, RS-232 and More
Source Level Debugging

See our full range of products, including Books, Accessories, and Components at:
www.melabs.com

Front Panels?

Unrivaled in price and quality for small orders

Download the free ›Front Panel Designer‹ to design your front panels in minutes



Order your front panels online and receive them just in time

www.frontpanelexpress.com

NEW

RS232 to TCP/IP

- *TCP/com*™ v2.0, RS232 to TCP/IP software. Plus TCP/IP to RS232.
- *WinWedge*™ RS232 or TCP/IP data direct into any Windows app. - Excel, Access, etc.

TALtech

Free 30 day evals at
www.taltech.com

CUSTOM MEMBRANE KEYBOARDS / SWITCHES

- 1 TO 2 WEEKS TURNAROUND
- VERY COMPETITIVE PRICING
- Ex.: (5) 4-switch keyboards for $395.00
- PCB backed switches
- Custom metal backplates/assemblies
- Electronic assemblies/graphic overlays
- Electronic file transfer capabilities

Picofab Inc.

4780B Blvd. Henri-Bourassa
Charlesbourg, Quebec, Canada G1H 3A7
Tel: (418) 622-5298 • Fax: (418) 622-9996
Email: sales@picofab.net

PRINTED CIRCUIT BOARDS

QUALITY PRODUCT
FAST DELIVERY
COMPETITIVE PRICING

PROTOTYPE through PRODUCTION

- UL approved
- Single & Double sided
- SMOBC
- LPI mask
- Through hole or SMT
- Nickel & Gold Plating
- Routing or scoring
- Electrical Testing
- Artwork or CAD data
- Fast Quotes
- Flex Circuits

SPECIAL OFFER:
10 pcs (3days) 1 or 2 layers $249
10 pcs (5days) 4 layers $695
(up to 30sq. in.ea.) includes tooling, artwork, L.P.I. mask & legend

PULSAR, INC

9901 W. Pacific Ave. Franklin Park, IL 60131 • Phone 847.233.0012
Fax 847.233.0013 • Modem 847.233.0014 • www.pulsar-inc.com • yogii@flash.net

Weeder Technologies

www.weedtech.com DATA SHEETS

RS-232 Stackable

1710-B Brighton Cove, Ft Walton Beach, FL 32547 850-863-5723

Digital I/O • Analog Input • Analog Output • Solid State Relays • Stepper Motor Controllers • Pulse Counters • Pulse Timers • Frequency Measurement • RPM Measurement • Duty Cycle Measurement • Period Measurement • Projectile Velocity Measurement • Third-Party Peripheral Interface • Data Logging

Plug end-to-end
Stack 32 modules on the same RS-232 cable.

Download Windows software

Data Acquisition & Control

...simple as a laptop.

NOW YOU CAN HAVE IT ALL!

- *Easy to Use*
- *Low Cost*
- *Low Power*
- *Tiny Size*
- *High Speed*
- *DOS Environment*

with the

DOS Stamp™

Easy Software Development: Use your C/C++ or Basic compiler to produce DOS EXE. Download EXE to flash disk via serial port. EXE runs on power up.

Standard Features: BIOS & DOS-ROM, 128K flash disk, 512K SRAM, 40 MHz CPU, 16 digital I/O (opto rack interface), 2 RS-232, 2 timer/counters, simple bus interface, real-time clock

Options: 8-ch 12-bit ADC, flash disks up to 288 MB, 1 RS-485, TCP/IP

Tiny Size, Low Power: 2"x2.6", 5V @ 200 mA at full speed, 40 mA at lowest speed.

Visit http://www.bagotronix.com for info, prices, and FAQs

BAGOTRONIX
2900-1 Crescent Drive
Tallahassee, FL 32301
850-942-7905 phone & fax

INDEX OF ADVERTISERS

The Index of Advertisers with links to their web sites is located at www.circuitcellar.com under the current issue.

Page		Page		Page		Page	
91	All Electronics Corp.	72	FaabMedia, Inc.	64	MaxStream	32	R.E. Smith
92	AP Circuits	86	FDI-Future Designs, Inc.	89	MCC (Micro Computer Control)	18, 29	Rabbit Semiconductor
27	Arcom	89	FieldServer Technologies	82	Micromint	66, 87	Rabbit Semiconductor
90	ASIX	93	Front Panel Express	92	microEngineering Labs, Inc.	9	RadioShack
7	Atmel	91	Futurlec	90	MJS Consulting	88	Reach
93	Bagotronix, Inc.	86	Grid Connect	91	Mosaic Industries, Inc.	50	Renesas
48	Bellin Dynamic Systems, Inc.	91	Hagstrom Electronics	42	Mouser Electronics	72, 89	Saelig Company Inc.
79	Beta Layout Ltd.	55	HI-TECH Software, LLC	49	MVS	3	Scott Edwards Electronics, Inc.
74	CadSoft Computer, Inc.	63, 89	ICOP Technology, Inc.	90	Mylydia, Inc.	88	Sealevel Systems
87	Carl's Electronics	89	IMAGEcraft	C2	NetBurner	56	Sensors Expo & Conf.
48	CCS, Inc.	90	Intec Automation, Inc.	15	Noritake	5	Sierra Proto Express
92	Conitec	88	Integrated Knowledge Systems	75	Nurve Networks LLC	25	Silicon Laboratories, Inc.
10	Custom Solutions	91	Intrepid Control Systems	88	OKW Electronics, Inc.	88	SMTH Circuits
65	CWAV	92	Intronics, Inc.	92	Ontrak Control Systems	10	SPLat Controls Pty Ltd
1	Cypress MicroSystems	11, 58	Jameco	2	PCBexpress	93	TAL Technologies
86	Decade Engineering	64, 92	JK microsystems, Inc.	87	PCB Fab Express	C3	Tech Tools
91	Digital Products Co.	75	JR Kerr Automation & Engineering	2	PCBpro	40, 41	Technologic Systems
86	DLP Design	73	Keil Software	C4	Parallax, Inc.	87	Technological Arts
81	Dynon Inst.	75	LabJack Corp.	21	Philips ARM Design Contest	89	Tern, Inc.
18	Earth Computer Technologies	86	Lakeview Research	86	Phytec America LLC	90	Trace Systems, Inc.
95	Echelon Corporation	90	Lawicel HB	87	Phyton, Inc.	91	Triangle Research Int'l, Inc.
87	EE Tools (Electronic Engineering Tools)	91	Lemon Studios	93	Picofab, Inc.	90	VNISource Design
90	Elsevier, Inc.	47	Lemos International	93	Pulsar, Inc.	93	Weeder Technologies
79	EMAC, Inc.	35	Linx Technologies	89	R2 Controls		
19	ExpressPCB	17	M16C Design Contest	57	R4 Systems, Inc.		

Preview of June Issue 179
Theme: Measurement & Sensors

Accurate Capacitance Meter

Precision Frequency Meter

Electronic Angle Measurement

Short-Range IR Communications System: Progressive Reflective Transmission Solution

Low-Cost Logic Analyzer for FPGAs

Intelligent Automatic Garage Door Opener: PIC and Sensors Sophisticate Automation

---

ABOVE THE GROUND PLANE RF Attenuation: Resistors at RF

APPLIED PCs Embedded USB Made Simple

FROM THE BENCH Surface-Mount Technology Guide

SILICON UPDATE Best in Show

ATTENTION ADVERTISERS

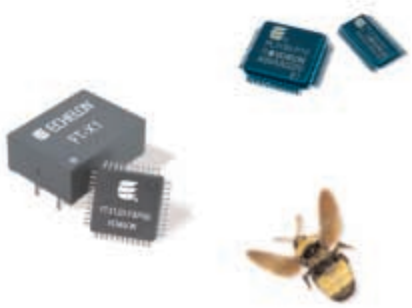
July Issue 180 Deadlines

Space Close: May 10
Material Close: May 17

Theme:
Internet & Connectivity

Call Sean Donnelly now to reserve your space!
860.872.3064
e-mail: sean@circuitcellar.com

High performance. Low cost. No sting!



Do you want to add networking to your embedded devices but keep getting stung by high product and development costs? Help is at hand. Echelon's embedded LONWORKS® platform is high on performance and low on product cost and development time. Our new Mini EVK gets you from design to delivery in no time, and at very little cost. With more than 40 million devices fielded, isn't it time you found out why so many embedded system developers choose Echelon. Call +1 408 938 5200, or visit us online at www.echelon.com/mini to get started today.

ECHELON®

©2005 Echelon. Echelon, LonWorks, and the Echelon logo are registered trademarks of Echelon Corporation registered in the United States and other countries.

PRIORITY INTERRUPT

by Steve Ciarcia, Founder and Editorial Director

Are You Driven by the Means or the End?

Hopefully you are among the people who frequent the *Circuit Cellar* Discussion Board forums (http://bbs.circuitcellar.com/phpBB2). Along with interesting handles, occasionally we get some thought-provoking questions, like these:

Does anyone feel like they want to get involved in competitions, submit articles, and stuff like that, but start with a means rather than an end? What do most people here do about their brainstorming sessions? Do they start with a means or an end? Any thoughts? –LucidGuppy

When I read this, I interpreted it as asking whether I think about the chicken or the egg first when I build a project. In truth, I'm guilty of all of the above. If I'm on a quest, I'll either use the fact that I already have everything to do a task (the means) or use my interest in solving a problem (the end) as justification to collect everything to do the task. Half of the products you saw come out of my early articles (the Home Control System (HCS), for example) came about because they sounded like fun to design and I wanted one for my own use. It didn't hurt that I had smart guys around here to fill in some of the blanks (like software).

I was able to produce these devices in months rather than years because we had a very large and eclectic inventory to satisfy my "interests." I've always been a bit of a pack rat. Having a large house and lots of garage space doesn't help people with this habit. Don't get me wrong, I'm not a junker. I'm not knee-deep in useless piles of electronic parts in place of the bed in the guest room. The tablecloth covering my end table isn't hiding a pile of old 8″ disk drives and equipment boxes. For the most part, everything is on shelves and cabinets either out behind the Circuit Cellar (and yes, newbies, there really is a Circuit Cellar) or in the garages. It's just that there is a lot of it.

Call it a personality defect or an excessive case of planning ahead, but if I get interested in an idea, I start collecting the means to accomplish it. If I start thinking about adding a few web cams, I start picking up cameras, more wire, wireless transmitters, Wi-Fi antenna extenders, etc. But, of course, I don't know exactly what kinds of problems I'll run into when I finally start installing, so I need to stock both color and black and white cameras, both regular and low light. You get the picture.

I'm one of those guys who doesn't like just talking about doing things. When I am ready, I want to do it right then. I don't want to wait six weeks for a back-ordered part, because the odds are that I'll probably be onto something else by the time it shows up. I generate lots of ideas, and I don't want to miss executing one when the time is opportune.

Of course, this runs completely counter to the rules. The typical engineering process involves iterative steps and lots of planning. And then finally, you start your project. In my mind, this is for wimps.;-) I prefer burst mode project execution. When I get an idea, I just keep it on the back burner while I collect every conceivable item to complete the task in the shortest possible time. I don't like putting a project aside to collect parts after I start working on it. Only when I have accumulated all of the potential ingredients do I really start examining the exact design solution. The downside to this method is that, when I have a dozen different ideas running in parallel or I don't use all of the hardware I've collected for each project, the "means" start piling up—big time.

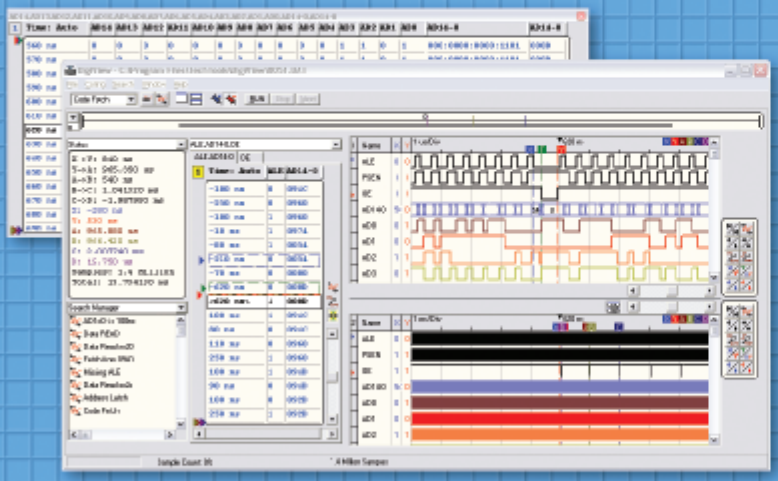
I've gotten better in recent years (after all, how much junk can you really collect?), but I still have a lot of stuff and too many projects I'd like to build. Old-timers who have been following *Circuit Cellar* since *BYTE* probably remember the famous tag line associated with my column. Someone once asked me what programming language I liked best. It's no secret that I'm a hardware guy, so I answered, "My favorite programming language is solder." It stuck, and lots of people still laugh when they hear it.

Well, there's another tag line for Steve Ciarcia if you know me personally or have ever been to my house: "Someplace I have everything!"

steve.ciarcia@circuitcellar.com

DigiView

100 MHz, 18 Channel, Portable Logic Analyzer!

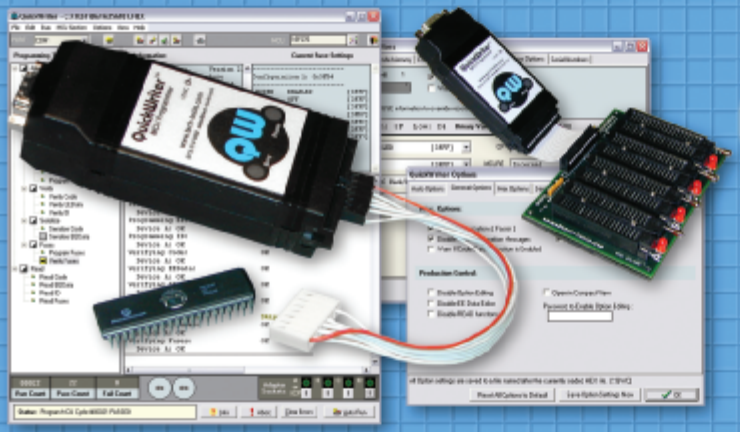


Auto Hardware Compression Captures from 128K to 30 Billion Samples @10ns resolution!

only $499.00

- Save up to 5 minutes of Data @ 10ns.
- Edge and Pattern Triggers.
- Real-time Hardware Compression.
- USB Powered for Portability.
- Includes Capture & Analysis Software.
- Pattern Searches with Match, Duration & Animation.
- Drag & Place and Drag & Snap Markers.
- Waveform Next/Previous/Nearest Edge snapping.
- Multi-Signal Data Tables with Link Groups.
- Split Waveform Views with Link Groups.

Download and try the Software!



QuickWriter

Multi-Function PICmicro® MCU Programmer In-Circuit & Gang Operation

only $199.00

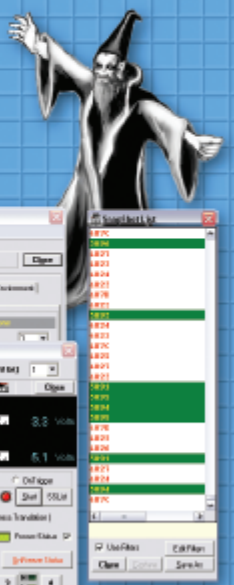
- Supports 12, 14, 16 & 18 Series PICmicro MCUs.
- Edit Code, EE Data, IDs and Configuration Fuses.
- Auto or Manual Serialization in Code or EE Data.
- Firmware auto synchronizes to Software version.
- Control Files protect Option Settings.
- Accepts Command-Line Parameters.
- Tests and Monitors Voltages.
- Control Signal for Self-Powered Circuits.
- Single and 4 Gang Adapters available.
- Includes Cables, Software & U.S. Power Supply.

See Our Other Embedded Development Tools

EconoROM III & FlexROM III

EPROM & FLASH Emulation at its best!

from $179.00



TechTools

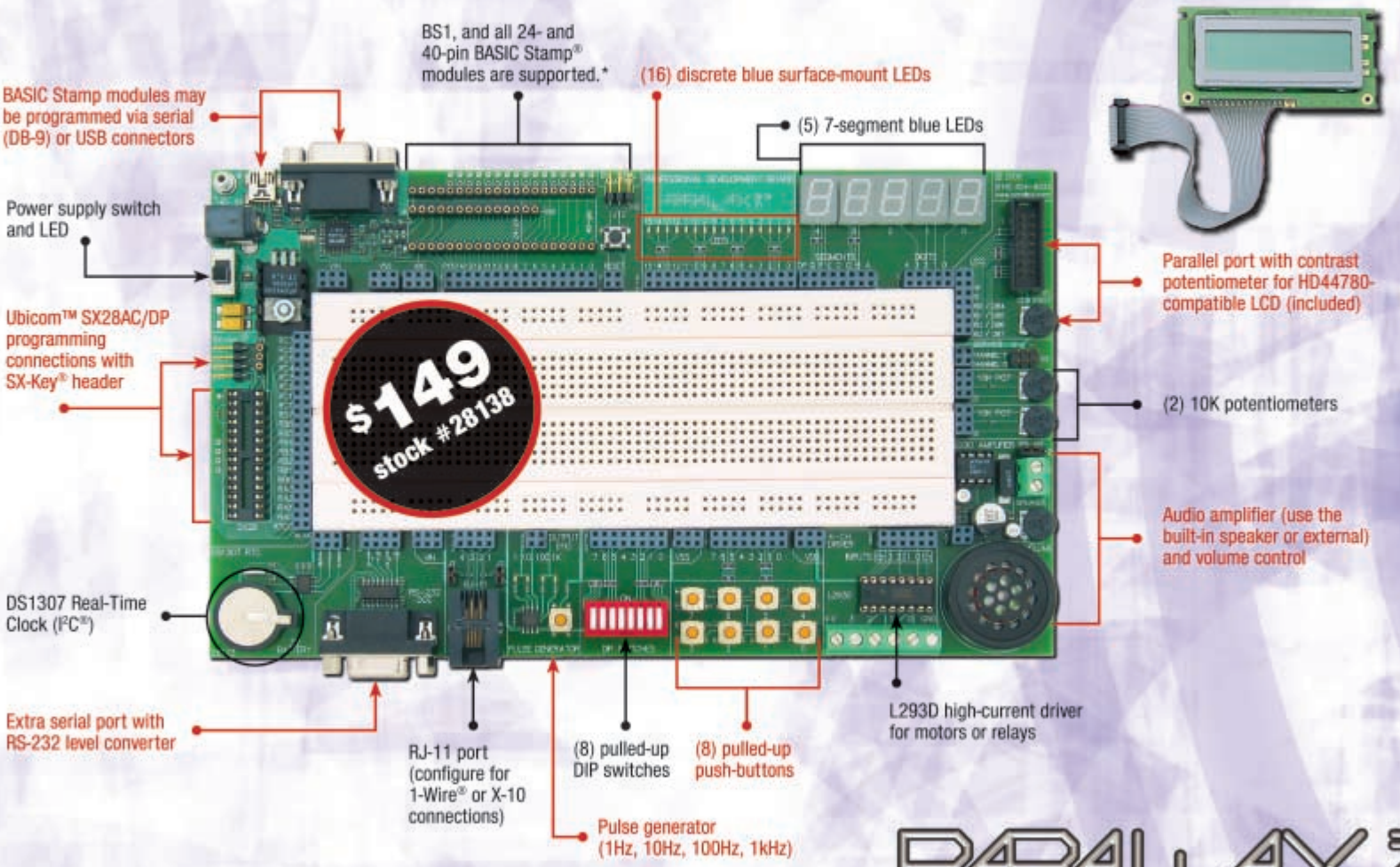
www.tech-tools.com

(972) 272-9392 • sales@tech-tools.com

Copyright © 2005 TechTools • DigiView, FlexROM, EconoROM and QuickWriter are trademarks of TechTools • PICmicro is a registered trademark of Microchip Technology Inc.

Parallax Professional Development Board

Program all BASIC Stamp® microcontrollers and the SX28AC/DP microcontroller with this feature-loaded development board. Popular I/O devices are built-in and a large breadboard provides plenty of prototype space.


BS1, and all 24- and 40-pin BASIC Stamp® modules are supported.*
(16) discrete blue surface-mount LEDs
(5) 7-segment blue LEDs
BASIC Stamp modules may be programmed via serial (DB-9) or USB connectors
Power supply switch and LED
Ubicom™ SX28AC/DP programming connections with SX-Key® header
$149
stock #28138
DS1307 Real-Time Clock (I²C®)
Extra serial port with RS-232 level converter
RJ-11 port (configure for 1-Wire® or X-10 connections)
(8) pulled-up DIP switches
(8) pulled-up push-buttons
Pulse generator (1Hz, 10Hz, 100Hz, 1kHz)
L293D high-current driver for motors or relays
Audio amplifier (use the built-in speaker or external) and volume control
(2) 10K potentiometers
Parallel port with contrast potentiometer for HD44780-compatible LCD (included)

* *Javelin Stamp is not supported.*

BASIC Stamp and SX-Key are registered trademarks of Parallax, Inc. Parallax and the Parallax logo are trademarks of Parallax, Inc. 1-Wire is a registered trademark of Dallas Semiconductor. I²C is a registered trademark of Philips Semiconductor. Ubicom is a trademark of Ubicom, Inc.

PARALLAX

www.parallax.com

Order online or call our Sales Department toll-free at 888-512-1024 (M-F, 7am-5pm, PT)